AUUG 2002

Measure • Monitor • Control

Conference Proceedings

4-6 September 2002
Duxton Hotel, Melbourne

http://www.auug.org.au/

AUUG Inc

AUUG Inc
PO Box 366,
Kensington NSW 2033
Australia

Phone: 1-800 625 655
or +61-2-8824 9511
Fax: +61-2-8824 9522
Email: auug2002@auug.org.au

Conference Proceedings

AUUG 2002
Measure, Monitor, Control

4-6 September 2002

DUXTON HOTEL
MELBOURNE

SPONSORS

AUUG Inc gratefully acknowledges the valuable and generous contribution given towards **AUUG 2002 - Measure, Monitor, Control** by the following organisations.

DIAMOND SPONSOR

http://developer.apple.com/

PLATINUM SPONSORS

http://www.checkpoint.com/

http://www.sun.com.au/partners/developer/

GOLD SPONSOR

http://www.ibm.com/

SILVER SPONSOR

http://www.samsungcontact.com

PRESS SPONSOR

https://secure.zdnet.com.au/technologyandbusiness/

PARTICIPATING SPONSORS

http://www.aarnet.edu.au/

http://www.caldera.com/

SUPPORT

http://www.pearsonED.com.au/

The Australian Computer Society (ACS) is the recognised association for Information Technology (IT) professionals, attracting a large and active membership (over 16,000) from all levels of the IT industry. It provides a wide range of services to its members. A member of the Australian Council of Professions, the ACS is the public voice of the IT profession, the guardian of professional ethics and standards in the IT industry, with a commitment to the wider community to ensure the beneficial use of IT. Visit **www.acs.org.au** for more information.

AUUG 2002 Conference and Programme Committee:

Adrian Close (Programme Chair)
Elizabeth Carroll (Conference Chair)

Frank Crawford
Peter Gray
Jason King
Greg Lehey
David Purdue
Con Zymaris

Editor: Adrian Close `<adrian@auug.org.au>`

ISBN: 0 9577532 4 1

PREFACE

The last year has seen a number of events conspiring to effect a general malaise on the IT industry – no-one has been unaffected by this. At the same time, exciting things are happening, not least of which is Apple's re-entry into the Unix space, bringing their graphic technology and usability know-how to an operating platform we know well and love.

Our conference theme of *Measure, Monitor, Control* seems to have inspired a much larger than usual number of submissions – easily enough for two conferences and all of good quality. This made the selection process rather a difficult one, but we think you'll agree the resulting programme is a solid one.

We welcome the participation of our overseas guests, including Rob Pike (Bell Labs), Alex Noordergraaf (Sun) and from Japan, Jun-ichiro "itojun" Hagino (KAME Project). Their input will help us keep a good perspective on what we're doing. Several wandering Aussies also return to fill out the picture, including Neil Gunther and John Terpstra.

Of course, the local scene continues to be a source of innovation, with participation in areas ranging from operating system development through large scale system management and cutting edge networking.

Once again we are pleased to have Kimberley Heitman present his commentary on some of the political and ethical meta-issues facing both us as computing professionals and the broader membership of our increasingly digital society.

We greatly appreciate the support of all our sponsors – without them the conference could not be what it is. It is heartening to note that major technology vendors are willing to assist our efforts in promoting the use of Unix and open systems.

This year's conference proceedings have been produced using the LaTeX typesetting system. A great deal of effort has gone into converting source documents from various formats, processing images, formatting and proof-reading. I hope the end result makes for printed output that is pleasing to the eye.

I wish to thank Malcolm Herbert, Neil Gunther, Martin Schwenke and members of the Programme Committee for their assistance with the production of this book. The legacy of Frank Crawford's previous years of hard work as Programme Chair has also been invaluable. Last but not least, I must thank my employer, Fernhill Technology, for their kind indulgence throughout the conference organisation process.

And now, without further ado, the Programme Committee and I commend to you the proceedings of the AUUG2002 Conference. As always, we hope that you will enjoy your reading and gain new insight into the bargain.

Adrian Close
Programme Chair
AUUG2002

Paper Selection Summary	
Submitted	52
Rejected	22
Withdrawn	2
Selected	28

Contents

Wednesday 4th September 2002

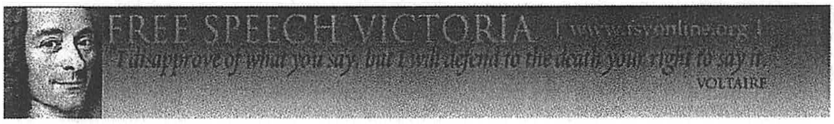
ICT and eGovernment

Dr Steve Hodgkinson
Director eGovernment Strategy & Policy
Multimedia Victoria
`http://www.mmv.vic.gov.au`

Dr Steve Hodgkinson will open the conference with a brief presentation on the Victorian State Government's eGovernment strategy and will outline the role of ICT in enabling transformation of Government and the key ICT issues which must be addressed.

Steve is the Director eGovernment Strategy & Policy at Multimedia Victoria - the unit of the Victorian State Government charged with leading implementation of the Government's Connecting Victoria policy. He is responsible for leadership of eGovernment strategies and activities across the Victorian Public Sector, including the promotion of best practices and communities and practice in IT and eServices.

Conversations on the Electronic Village Green – protecting freedom of speech in cyberspace



Terry Lane
President, Free Speech Victoria
`http://www.fsvonline.org`

Terry will talk about a number of recent cases which have tested the limits of free speech on the Internet – including examples which have involved defamation, racial and religious vilification and obscenity.

He will examine the nature of the Internet and compare it with other communications media to see how it is similar to and how it differs from newspaper, radio, television, film etc. He argues that those who value the Internet as an unmediated medium will have to fight to protect it from the censors who want to control it, as they presently control other communications media.

The Power of Unix – the Simplicity of the Mac

John Zornig
Systems Engineer
Apple Computer

History

"Apple ignited the personal computer revolution in the 1970s with the Apple II and reinvented the personal computer in the 1980s with the Macintosh."

The reinvention in this quote was the Mac graphical user interface (GUI) that introduced a new way of interacting with a computer. The computer in question, the Macintosh, had only 128KB RAM and a 400KB floppy drive for storage and a small non-colour bitmapped screen. The Motorola 68K CPU had no memory management and the Mac OS, as it has become known only ran one application at a time.

Warp through time to today and my PowerBook G4 still ships with an OS, Mac OS 9, that is the direct descendant of that first Mac OS. However this PowerBook is a very different beast to the original Mac. It is equipped with an 800Mhz G4 PPC with Velocity Engine, i.e. 128bit vector processor, 512MB RAM, 40GB Hard Drive, Gigabit Ethernet, FireWire, Wireless networking, 3D Accelerated high-resolution colour screen, ...

Apple realized many years ago that this could not go on forever. The Mac OS needed a rewrite to put it on a more solid foundation with intrinsic support for many of the technologies that were being retrofitted into Mac OS 9. Apple announced its acquisition of Next Software Inc. in December 1996 and the development of what is now Mac OS X "ten" began.

Architecture

Designing a new commercial operating system is not a small task and Apple took to heart the concept that things "not invented here" could be just as good as those created in-house. One of the base principles behind the architecture of Mac OS X is to build upon a foundation of existing technologies. But to do this successfully the big picture of what the OS needed to achieve had to be kept clear. The Mac OS X Architecture diagram is part of this big picture. It conveys the modular and layered nature of the Mac OS X architecture.

This doesn't look like other UNIX-Based systems. That is because the user environment for a Mac user is supported by Unix but removed from it by layers of graphical and application support technologies. Don't worry, the Unix is still there just the way you like it!

Darwin

Let's start at the bottom and look at Darwin. Mac OS X is a Unix-based OS and Darwin is the core of Mac OS X and where the Unix is located. Darwin is a monolithic kernel built from Mach and BSD UNIX with an architecture for dynamically loadable kernel modules for file systems, networking, and drivers. Darwin's user interface is the familiar Unix command line with pretty much everything you would find on other Unix-based platforms. However

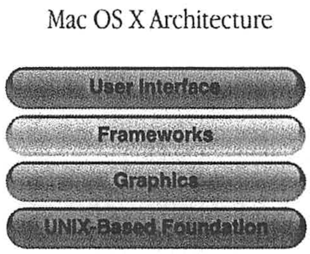


Figure 1: Mac OS X Architecture

the way it is designed provides capabilities in areas like multiprocessing, memory management, low-latency process scheduling and power management that straight BSD or most other traditional Unix-based systems can't match.

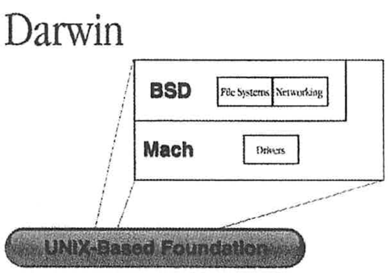


Figure 2: Darwin

Let's have a quick look through the other layers of Mac OS X before returning for a closer look at Darwin.

Quartz

The graphics layer in Mac OS X is called Quartz. This is based on PDF for 2-D graphics, Open-GL for 3-D and QuickTime for 4-D or multimedia. These industry standard graphics systems provide a rich programming environment for developers and because they are all integrated together, the results for the user, as you will see, are stunning.

Framework

Mac OS X packs a lot into this layer, because it is here that developers can leverage existing technology to their advantage without having to reinvent the wheel in each new project.

The traditional BSD APIs are supported of course, providing a pretty good coverage of POSIX. Sun's Java 2 SE is also fully integrated into this layer, supporting pure JAVA applications. Apple has built an application environment that supports two API's, Carbon and Cocoa. Carbon is a procedural API derived from the traditional Mac OS toolkit. Cocoa is an Object Oriented API derived from the NeXT OpenStep environment. Cocoa APIs are available to both Java and Objective-C programmers. The final application environment - Classic, encapsulates a copy of Mac OS 9 and any Mac OS 9 applications within a single multi-threaded task inside Mac OS X. Applications in all of these environments are equal in the eyes of the underlying Darwin layer.

Aqua

One of the wonders of the Aqua user interface is the seamless integration of applications from these different environments. Aqua is instantly familiar to a Mac OS 9 user but the interface has been completely rethought and reimplemented for Mac OS X. What was under Mac OS 9 a monolithic and sparsely threaded process is under Mac OS X a number of independent highly multi-threaded tasks that make use of the capabilities of the underlying layers to the full.

Darwin & Open Source

Returning to Darwin, one concept to remember is that Darwin is an OS in it's own right, independent from Mac OS X. Apple has made the Darwin layer open source and there is an active community of over 100,000 developers accessing the live Darwin source code repository at developer.apple.com/darwin/. Recently Apple has also cut Darwin loose and it has it's own website, OpenDarwin.org, and repository outside Apple and controlled by the developer community. The CVS at Apple is live and is updated as engineers at Apple and selected members of the community "Darwin Committers" make changes. Apple also makes binary installers for Darwin on PPC and Intel architectures available for download.

Darwin in Mac OS X includes a wide range of software from the open source world and much more is available from the Darwin CVS or the official sites of individual projects. The compatibility between Darwin and other Unix is high, so that in the short time since the release of Mac OS X a vast array of software has been ported. Most major open source projects now list Darwin/Mac OS X as one of their supported platforms. Sites like Sourceforge include Mac OS X in their build farms, encouraging continued development.

Darwin and the GUI

The default Unix GUI is X-Windows and Mac OS X has Aqua. This dichotomy has not slowed the rush of Unix software to Mac OS X. Quite the opposite. X-Windows is ported to Darwin OS and though it is not shipped with Mac OS X it is easily obtainable and integrates so well with Aqua that it never fails to take people by surprise when they first see it. X-Windows and Aqua are very different animals and the lure of an easily programmed sophisticated GUI is tempting many developers to develop new Aqua UIs for their apps.

Developer Tools

Darwin's developer tools are those of the open source community, gcc, make, Java etc. Mac OS X's developer tools build a complete GUI integrated development environment above those tools. Project Builder and Interface Builder are the primary tools and come free with Mac OS X. A developer equipped with these tools can program for the kernel, for BSD, pure Java, Cocoa, Carbon, and even AppleScript, the Mac OS system wide scripting language.

There is also a rich set of third party development tools, both commercial and open source.

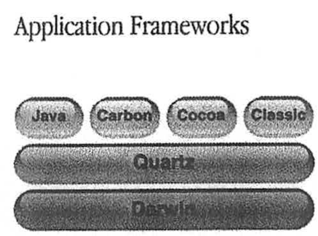


Figure 3: Application Frameworks

Incredible Hardware

Mac OS X allows us to bring UNIX to what are widely regarded as the best-designed products on the planet. Because Apple develops both the hardware and the software, the result is tight integration and great plug-and-play, as well as full support of modern standards like Wireless, USB and Firewire. This is the hardware many people in the UNIX community have been lusting after—and now they can enjoy it "guilt-free", since they're still running an Open Source UNIX-based operating system.

The Dream Machine

Unix has a long history and throughout that history Unix users have dreamt of, and strived to develop, a Unix system with the features and capabilities that Mac OS X delivers today. Now that the dreamers have their machine, Unix has a new future.

References

`http://www.apple.com/macosx`
`http://developer.apple.com`
`http://lists.apple.com`
`http://www.opendarwin.org`
`http://fink.sourceforge.net`
`http://osx.hyperjeff.net/links.html`

'Two years in the trenches': Evolution of a free software project

Greg Lehey
President, AUUG Inc.
Greg.Lehey@auug.org.au
Member, FreeBSD Core team
grog@FreeBSD.org

Abstract

Free software projects have been around for over ten years now, and they've become an important part of the UNIX community. With the example of the FreeBSD project, this paper illustrates some of the changes that have occurred in that time and attempts to guess what the future may bring.

The three ages of UNIX

UNIX® is now a third of a century old. In this time, the face of computing has changed dramatically. It's interesting to consider the evolution of UNIX in three phases of roughly 11 years each:

- From 1969 until about 1980, UNIX was mainly a research project, little known outside AT&T except at some universities.
- From 1980 until about 1991, UNIX developed into a commercial operating system with releases like UNIX System V (remember that?), XENIX, SunOS, Ultrix and friends.
- In the early 90s, efforts at the University of California in Berkeley to produce free UNIX came to fruition. At pretty much exactly the same time, the Linux project started. UNIX was on the way to becoming free.

This paper looks at the last third of this history in more detail.

The evolution of free UNIX

Free software has been around for ever. Until the end of the 60s, coincidentally the time when UNIX evolved, software was almost always free. And why not? Most system software came with the machines and would only run on them, so there was little incentive to charge separately for the software. Applications software was tuned to specific applications, so piracy was hardly worth worrying about.

By the end of the 1960s, that had changed. IBM was faced with competitors who built hardware which was strikingly compatible with the System/360, and IBM's software ran on it. It started to make commercial sense for IBM to unbundle its software. Over the course of the following ten years, more and more compatible hardware became available, and by 1980 just about all software cost money. Vendors had a perception that access to the source code would give their competitors an advantage, so they restricted availability of their software to object form only. The resultant inflexibility was one of the reasons which led to the formation of the Free Software Foundation in the mid-80s.

Towards the end of the 70s, the price of "real" computers dropped to the point where individuals could own them. People started sharing programs for their CP/M based machines, and later for their Microsoft-based replacements. Most software for small machines continued to cost money, however, and the machines themselves were too small to run UNIX effectively.

This changed round about 1990 with the availability of machines based on the Intel 80386 processor. At the same time, individual access to the Internet became easier, especially for students. These were the background for a number of free software projects which were initially completely independent of the Free Software Foundation and its values.

The early 90s

Computer hackers[1] have been around as long as computers. They have probably dreamed of having their own computer that long, too, but it really only became even remotely possible in the mid-1970s. People started working on UNIX-like systems early:

- In the late 1970s, a company called Electrolabs started working on a UNIX-like system called OS-2, intended to run on the Z-80 processor. It never passed the beta stage running under CP/M.
- In the early 1980s, Mark Williams Company ported their Coherent system, a clone of the UNIX Seventh edition, to the IBM PC. Without memory protection, and with the memory limitations of the original success, it was not a great success.
- Andy Tanenbaum's Minix operating system ran on a number of processors, including the Intel 8086 family.

None of these projects became very large. The hardware wasn't up to it, and none of them made the source code freely available, though the restrictions on Minix were relatively minor. Two things changed that situation in the late 1980s and early 1990s:

- The release of the Intel 80386 processor and systems based on it, gave the average hacker an affordable machine with virtual memory, capable of running modern UNIX without significant compromises.
- Improved access to the Internet, especially for students, made cooperation on software projects more practicable.

In those days, for most people the attraction was the challenge of running UNIX on one's own computer, not any commercial intent. Indeed, Bill Jolitz staunchly refused to commercialize his 386BSD operating system, and in August 1991, Linus Torvalds wrote in the original announcement of Linux "just a hobby, won't be big and professional like gnu".

I don't believe that religious belief in "free software" was a big thing at the time. The GNU project had been around for a while, and both 386 BSD and Linux took advantage of the software, but initially there was little synergy between the GNU project and the free OS projects.

During this period, relatively few people contributed to the projects. For example, the release notes of FreeBSD 2.0, released in January 1995, mention a total of 55 contributors, including a 15 member core team, who really *were* the project. Most of the other contributors had only loose links with the FreeBSD project.

The compromise OS

Along with the free operating system projects there was also a "reasonable" operating system project, BSD/386 from Berkeley Software Design Inc., or BSDI (later BSDi). BSD/386 was derived from the same code base as 386BSD, and was in fact quite closely related. The story of that relationship is interesting enough to be the topic for another paper. The difference was that BSD/386 cost money. Including complete source code, it cost $1,000 US. That may seem to be a lot of money, but in those days source licenses for System V cost at least $50,000, and a BSD/386 source license was was cheaper than a binary license for the System V implementations for the Intel platform. In addition, it was complete–dozens of additional packages to add, each with their own activation key. Even more surprisingly, BSD/386 was more reliable than System V.

The mid-90s

In the early days, nobody had very great expectations for the free operating systems. They didn't expect them to be as reliable as UNIX®System V. They didn't even expect them to be as reliable as Microsoft's offerings, though they did expect them to be easier to use.

In a few years, something happened that nobody had expected. Well, very few people: the operating systems did become as reliable as Microsoft's offerings, quite quickly in fact. One of the first results of this recognition came from the massive increase in Internet usage, which created a requirement for low-cost web servers. This was

[1] A hacker is, of course, somebody who can't tell the difference between work and play with computers. See the New Hacker's Dictionary http://www.tuxedo.org/ esr/jargon/html/entry/hacker.html for more details.

the reason for the foundation of BSDI, who branded their operating system "Internet Server", later "Internet Super Server". Initially they were very successful. As time went on, though, enterprising startups realized that they could save the cost of the software by using free operating systems instead of BSD/386. Thus in 1995, when Yahoo! started up, it was based on FreeBSD.

Linux was slower to reach reliability than the BSD systems: it had to be written from scratch, whereas the free BSD systems were based on a code base at least ten years old, including the most mature of TCP/IP stacks.

By the late 1990s, though, Linux had effectively caught up with the BSDs, and depending on which bigot you ask, either overtook the BSDs or never quite made it.

In the mid-1990s, another thing happened: the general public became aware of the concept of free operating systems. They were still considered very much the realm of geeks, but they were becoming known.

During this time, more and more people became involved with the projects.

The late 90s

By the end of the 1990s, free software was well enough known in the IT industry for some people to form companies to market it. The result was a further increase in the size of the projects. At the time of writing, the FreeBSD project has 318 committers, in other words developers with write access to the source tree, probably ten times the number as of January 1995.

Social changes

Clearly the social structure of the FreeBSD project has changed greatly in its nature in the last ten years. By mid-2000, the strain was beginning to show:

- The core team was no longer the group of the most active comitters. Their function had effectively become more administrative, but they hadn't recognized the fact yet.
- The architectural direction of the project had become a little vague. The position of chief architect, previously held by David Greenman, had been vacant for some time. In the early stages of the project, most of the work in the project had been to make FreeBSD a stable UNIX-like operating system, but that goal had now been achieved. There was more work to do, but the only clearly defined goal was to improve the SMP performance–the SMPng project, which at the time had just started, and which is still continuing.
- A related problem was the attractiveness to end users. Like other free operating systems, FreeBSD has always been a developer-driven project, but the main source of project funding came from selling CD-ROMs. The sales of the CD-ROMs were obviously dependent on the perception of the purchaser, but nobody in the project was overly concerned with this issue.
- Some developers exploited the ineffectiveness of the core team by doing whatever they wanted. This notably included making changes to the system to match their view of what was needed, possibly breaking parts of the system (those parts which they didn't use), and leaving it to others to clean up.
- This in turn, along with an observed inability on the part of the core team to solve the problem, caused a serious decline in the morale of the project, and a number of people left the project as a result.
- One of the most noticeable rogue developers was a member of the core team. As a result, the lack of activity of the core team was perceived as cronyism.
- The core team had adopted a policy of not communicating the reasons behind its decisions, partly to hide the fact that many members were inactive, and partly to avoid sparking conflicts. Not surprisingly, this gave them a reputation for secrecy.

In November 1999, Nate Williams asked in a mail message:

> *Finally, what is the purpose of core? I used to know, but I'm not sure anymore. What determines if someone should become a core member? Is there any way to lose your core member status, in the same manner that you can lose the ability to be considered a maintainer? Do you have to quit in order to not become a core member? (So far that's the case).*
>
> *My *biggest* fear is that we will lose active developers simply because we just plod along hoping that everything will work out, and hope that someone will pick up the torch from time to time and take us in some sort of good direction.*

> *Lah-de-dah, lah-de-dah. Once upon a time, core members were folks were *really* excited and highly motivated to work on this thing, and would spend nights/weekends and all sorts of time on this. But, core is now older, and our real lives get in the way now.*

A couple of people suggested various ways to reform or change the core team, including the possibility of disbanding core altogether and becoming an anarchy, or voting for the core team. A number of people came up with remarkably complicated voting models. Finally, Jordan Hubbard came up with a suggestion and asked the developers to vote on a number of alternatives. Out of 94 votes cast, the most popular were:

- The idea of core is fine, its membership simply needs a shake-up and some mechanism added for voting in new blood. This alternative received 58 votes.
- The idea of core is fine, but some of members simply need to leave. This received 12 votes, most of which identified a single specific member.
- Core needs to be broken up into an oversight/human resources group, leaving architectural decisions to developers. This alternative received 9 votes.
- Don't change anything, core is fine the way it is. Received 7 votes
- Disband core entirely and let committers create a new structure in its place. Received 7 votes

Clearly the majority was for a democratically elected core team. More discussions ensued. Some people were concerned that politics would take over from reason, and the people who would get elected would be those who could drum up enough followers, not those who could do the best jobs. A team of volunteers, consisting of Jonathan Lemon, Warner Losh and Wes Peters, got together and thrashed out the existing voting models and came up with the following proposal, on which we also voted:

- Active committers have made a commit to the tree in the last 12 months.
- Core consists of 9 elected active committers.
- Core elections are held every 2 years, first time September 2000.
- Core members and committers may be ejected by a 2/3 vote of core.
- If the size of core falls below 7, an early election is held.
- A petition of 1/3 of active committers can trigger an early election.
- Elections will be run as follows: Core appoints and announces someone to run the election. 1 week to tally active committers wishing to run for core. 4 weeks for the actual vote 1 week to tally and post the results. Each active committer may vote once in support of up to nine nominees. New core team becomes effective 1 week after the results are posted. Voting ties decided by unambiguously elected new core members.
- These rules can be changed by a 2/3 majority of committers if at least 50of active committers cast their vote.

These "bylaws" passed by 117 yes votes to 5 no votes, thus also disproving the concern that committers wouldn't be interested enough to vote for the core team.

A couple of these provisions look a little unusual:

- The rationale behind the surprisingly long election period was that, since this is a volunteer project, many people might miss a shorter election period, especially Europeans on multi-week leave.
- We spent a lot of time discussing how to vote. We were concerned that if each voter had only one vote, the majority would vote for the same two or three candidates, effectively leaving the remainder to chance. Even worse, it could lead to less than 9 candidates being voted for at all. Initially, we also discussed a "veto" vote: "Don't let *that bloke* onto core". You'll note from Jordan's poll that the second most popular model was "expel *that bloke* from core". Neither of these suggestions were accepted.

The election

Nominations for candidacy were accepted between 5 September 2000 and 12 September 2000, after which the election started immediately. It finished on 10 October 2000. The results were announced on 12 October 2000, just in time for the beginning of BSDCon 2000 (http://www.bsdcon.com).

A total of 17 candidates registered, surprisingly close to the size of the previous core team. Only 8 of the previous core team stood for election. Campaigning was almost non-existent.

The morning after

The members of the new team, later to be dubbed *core.2*, were:

- Satoshi Asami, member of the old core team. Guardian of the Ports Collection Japanese.
- David Greenman, one of the founders of the FreeBSD project, and member of the old core team. Kernel hacker and former principal architect of the FreeBSD project. American.
- Jordan Hubbard, one of the founders of the FreeBSD project, and member of the old core team. Release engineer and former president of the FreeBSD project, a position which he had dropped some time before. American.
- Greg Lehey, newly elected. Kernel hacker, author of the Vinum Volume manager. Australian (Adelaide).
- Warner Losh, newly elected. Network hacker. American.
- Doug Rabson, member of the old core team. Kernel hacker, responsible for the port of FreeBSD to the Alpha platform. British.
- Mike Smith, newly elected. Low-level kernel hacker. Australian (Adelaide).
- Robert Watson, newly elected. Network hacker, FreeBSD security officer. British.
- Peter Wemm, member of the old core team. Universal Kernel Hacker. Australian (Perth).

In summary, the new team included five members from the old core team. Two candidates from the old core team were not re-elected. The composition of the team changed in other ways: five members of the old core team had a non-English native language, but only one member of the new team did. Seven members of the old core team lived outside the USA, only two of the new team did. Three members of the new core team were Australians, including myself, compared to two before.

One thing that all members had in common was that they were software developers, not managers. This is not surprising, given the mode of election. I had had some management experience years before, but I believe that I was the only one, and my experience wasn't much to write home about.

Into the trenches

The new core team took office at a panel discussion during BSDCon 2000. We had a completely new concept ahead of us, and we certainly weren't sure how to fulfil our objectives. Worse, we didn't even know what our objectives were. An association like AUUG has a constitution. The best we had in the FreeBSD project were the "bylaws", originally intended to define the modality of the elections.

The first meeting

The second FreeBSD core team had a meeting at the end of the BSDCon in Monterey. It was, in fact, a significant event: in the course of the history of the FreeBSD project, it was the only meeting of the entire core team. Previously, an attempt to pay to bring the core team together (at the FreeBSDCon in Berkeley, the year before) had failed thanks to the efforts of the US Immigration Department: Andrey Chernov, who lives in Russia, was deemed unsuitable for entry to the USA.

In this meeting, we tried to define what the purpose of the FreeBSD core team was. We discovered a surprising number of differences of opinion, but we finally decided:

- The FreeBSD core team does not define the architectural direction of the project.
- There will be no officers on the core team. Jordan Hubbard had suggested to take the role of spokesman, but the consensus was that people already saw him as exercising too much control on the team, and we suspected this would send the wrong message.
- The FreeBSD project is a volunteer organization, so the core team does not have a mandate to tell anybody to do anything.

That's conceding a lot. So what was left?

- The core team decides who can join the project as a "committer", somebody with commit access to the CVS tree. On request, backed by a *mentor*, who must already be a committer, the core team decides whether to admit him to the project (to "give him a commit bit"). Core voted on each application. A single "no" was sufficient to veto an application, and voting terminated after a week.
- In case of extreme misbehaviour, the core team can expel a committer from the project.
- In case of dispute between two committers, the core team mediates.

A little later we added the concept of a monthly core team report to address the accusations of secrecy made against the previous core team.
Comparing this list with the problems facing the project, a number of issues remained unanswered:

- We still had no architectural direction. The core team's intention here was that a consensus should be formed on the *FreeBSD-arch* mailing list. If no consensus could be formed, core would mediate.
- Attractiveness to end users. The majority of the members of the core team, being developers themselves, were not very interested in this aspect.
- Rogue developers. We couldn't agree on how to handle this one. One of the issues that was made very clear was that the core team did not have a "big stick". About the only thing that it could do would be to expel a member from the project.
- Project morale. This included behaviour of developers towards each other. Again, core did not come up with a good solution for this problem, though theoretically expulsion from the project would have been a solution.

Acceptance of core.2

How did we go? Parts of it were excellent.

The biggest problem we found was that core members were *still* unresponsive. Applications for commit bits, our main activity, took up to several months to process instead of the one-week timeout we had set ourselves. One of the problems was the amorphous structure of core: nobody had a particular hat, so there was nobody designated to actually convey the message back to the applicants. In the course of time, that got better. Very few applications were rejected.

Publishing the monthly core reports became very slow. Although the reports for the last months of 2000 were relatively timely, the January 2001 report was released on 29 June 2001. We began to recognize that we needed help, and solicited applications for the position of core secretary, a non-voting position more akin to AUUG's business manager than to the secretary. Initially we let several applicants work on the backed-up reports. The February 2001 report appeared in November 2001, and gradually we caught up with the backlog. We signed up Wilko Bulte as core secretary, a position he still holds, and by May 2002 we had cleared up the backlog.

Rogue developers

Not surprisingly, problems with rogue developers did not abate. Each occurrence caused a lot of angry discussion with core, which was very wearing on a number of the members. Surprisingly, it also became apparent that many core members saw each occurrence as a separate issue, and personal likes and dislikes were very evident. There was strong resistance to any general policy.

In February 2002, a developer announced his intention to commit some significant changes to the SMP code. At the time, the most active SMP developer, John Baldwin, was moving house from one coast of the US to the other, and was thus offline. Others who were involved pointed out that these changes were in conflict with changes that John was currently testing and asked the developer to hold off. The developer committed the changes anyway.

The handling of this particular issue became a test of core's authority. For the first time, core decided to revoke the developer's commit privileges if he did not back out the commits. He did so in the nick of time (without knowing about the impending suspension), and asked core to resolve the issue. The resolution was hard, and it looked more like tactics rather than strategy. After a month of discussion involving hundreds of mail messages, core appointed John Baldwin to the position of technical lead for the SMP project, with the power to approve or reject changes.

Based on this relative success, core deliberated and came up with some rules about developer behaviour:

- *Any committer who commits to the stable branch during a code freeze will have his or her commit bit suspended for 2 days. Any member of core or the re@ team can implement the suspension without the need for a formal vote within core or re@ respectively. The suspension will be published on -developers.*

- *Any committer who commits to the security branch without approval from the security-officer team will have his or her commit bit suspended for 2 days. Any member of core or the security-officer@ team can implement the suspension without the need for a formal vote within core or security-officer@ respectively. The suspension will be published on -developers.*
- *When committers engage in a commit war, both parties will have their commit bits suspended for 5 days. Any member of core can do this without the need for a formal vote. The suspension will be published on -developers.*
- *Any committer observed to act or speak in a way that is in conflict with the normal rules of interpersonal politeness, or in conflict with the best interests of the FreeBSD Project will have his or her commit bit suspended for 5 days. Any member of core can implement the suspension without the need for a formal vote within core. The suspension will be published on -developers.*
- *Core reserves the right to impose harsher penalties for repeat offenders. Harsher penalties include longer suspension terms and the permanent removal of commit privileges and FreeBSD.org accounts. Implementing harsher penalties are subject to a formal 2/3 majority vote in core. The outcome of core's decision will be published on -developers.*

In all cases where an individual FreeBSD officer takes a personal action he or she will be answerable to core. All cases can be taken for appeal to the core team. The outcome of such an appeal will be published on -developers and in the core monthly report.

These rules looked rather rigid, but we couldn't come to an agreement to moderate them, so that's the way they remain.

The big stick

In June, core received a formal complaint about the same committer who had caused us so much grief in February. He had committed code in an area on which another developer was working, without discussing the matter with him. This had annoyed the other developer to the point that he relinquished the maintainership of this part of the tree.

We discussed the matter and attempted to decide whether this behaviour was in conflict with the rules we had just published, specifically rule 4. A majority decided that it was, but there were extenuating circumstances. According to the rules, though, we still had to impose the full five day penalty.

The developers reacted in different ways, mainly unfavourably. In the meantime, as described the next section, we were in an election campaign. Some suspected that this punishment was politically motivated, since the developer in question was also an election candidate. It's not clear, however, whether this punishment improved or lessened his chances, but in any case he wasn't elected. The core team decided on a reprieve after two days, and the matter died down relatively quickly.

A few weeks later, two other highly respected committers engaged in a commit war: one committed something that the other didn't like, the other backed it out, the first recommitted it, and so on–a clear violation of rule 3. As always, there were extenuating circumstances. After some deliberation, core decided to suspend the commit bits for 24 hours. Again, this caused a commotion in the mailing lists, but it died down more quickly.

Is this working? It seems to. Core needs to understand how to dose the punishment, but the real issue here is not the temporary loss of commit privileges, it's the open recognition of inappropriate behaviour. It's still too soon to be certain, but maybe people are being more considerate as a result.

The collapse of core.2

Round May 2001, Satoshi Asami became sick and disappeared from the scene for some time. Even after his return, he did not participate in core discussions, and after several months, we finally decided that he was *de facto* no longer a member of the core team. According to the "bylaws", we carried on with only eight members.

After the SMP commit war described above, people were feeling tired. Everything seemed to take more effort than necessary. On 29 April 2002, Jordan Hubbard dropped a bombshell: he resigned from core. In his resignation message, he stated:

> *... being in core is honestly not what it once was. For an old-timer like myself, who was used to a core team that was far more cohesive and generally on the same page, it's simply a painful experience a lot of the time. Perhaps this is due to overly rose-colored recollections of the old core on my part, and I do certainly recall us having more than our share of disagreement and inefficiency in the past, but on the balance core still feels too much like the pre-WWII Polish Parliment sometimes, where we're fully capable of arguing some issue right up to the point where tanks are rolling through the front door and rendering the whole debate somewhat moot. I'm also not blaming this on the democratic model we've*

adopted, a stance which would be hypocritical at best since I'm one of the folks who argued strongly in favor of it, but I guess it's going to take a few more iterations before we get it right. It will also probably be a lot easier for truly new people who don't have a lot of preconceived notions of what core is to make that happen.

Finally, it also bears noting that while being part of the FreeBSD project is many things, it should always be "fun" to at least some degree for its participants or there's really not much point in being involved. Being in core, where one gets to deal almost solely with conflict resolution and bureaucracy, is not fun in any sense of the word and while being in core constitutes the bulk of my involvement, without any cool development work (which I also haven't had time for) to counter-balance it, it simply leaves me with less and less enthusiasm for FreeBSD.

Yes, it certainly wasn't fun. Slashdot picked it up with glee, of course, and the usual "FreeBSD is dying" trolls came out again. That's not the point, of course. This has nothing to do with FreeBSD as an operating system, these are simply some interesting project dynamics.

As if that wasn't interesting enough, five days later Mike Smith also resigned from core. In his message, he wrote:

FreeBSD used to be fun. It used to be about doing things the right way. It used to be something that you could sink your teeth into when the mundane chores of programming for a living got you down. It was something cool and exciting; a way to spend your spare time on an endeavour you loved that was at the same time wholesome and worthwhile.

It's not anymore. It's about bylaws and committees and reports and milestones, telling others what to do and doing what you're told. It's about who can rant the longest or shout the loudest or mislead the most people into a bloc in order to legitimise doing what they think is best. Individuals notwithstanding, the project as a whole has lost track of where it's going, and has instead become obsessed with process and mechanics.

...

From a technical perspective, the project faces a set of challenges that significantly outstrip our ability to deliver. Some of the resources that we need to address these challenges are tied up in the fruitless metadiscussions that have raged since we made the mistake of electing officers. Others have left in disgust, or been driven out by the culture of abuse and distraction that has grown up since then. More may well remain available to recruitment, but while the project is busy infighting our chances for successful outreach are sorely diminished.

Does this look familiar? It should do to anybody in commercial software development projects. Coincidentally, both Mike and Jordan work for Apple. They must be involved in project planning there; it's to be expected that it works better at Apple.

core.3

After Mike Smith's resignation, core only had six members left. According to the "bylaws", this meant that elections had to be held. That, too, caused long discussions. When, how quickly, should we change the "bylaws" first? We did a straw poll which showed that the committers did not want to wait, and they didn't want to change the bylaws. An election schedule was published in accordance with the "bylaws", and then people decided there wasn't enough time for people to declare their candidacy. Accordingly, the period for nominations was extended, and voting ran from 29 May to 25 June.

Despite the perceived tiredness, a record number of nominations were received. In contrast with the first core elections, there was a significant amount of politicking, with some candidates publishing lists of their preferred partners in core.

The results were announced, not as planned on 1 July, but immediately after the elections closed. The third core team consists of the following members:

- John Baldwin, newly elected. FreeBSD SMP technical lead. American.
- Jun Kuriyama, newly elected. Japanese.
- Greg Lehey, member of core.2. Kernel hacker, author of the Vinum Volume manager. Australian (Adelaide).
- Warner Losh, member of core.2. Network hacker. American.

- Mark Murray, newly elected. Security hacker. Zimbabwean.
- Wes Peters, newly elected. Network hacker. American.
- Murray Stokely, newly elected. FreeBSD Release Engineer. American.
- Robert Watson, member of core.2. Network hacker. British.
- Peter Wemm, only member of the original core team left. Universal Kernel Hacker. Australian (Perth).

This time around, all those members of the second core team (four) who stood for election were re-elected.

In many ways, this core team composition is the best we have had. In particular, we have better technical representation (John Baldwin) and the representation of the Release Engineer, Murray Stokely.

The first elected FreeBSD core team got off to an erratic start. As the resignation letters of Jordan Hubbard and Mike Smith show, this was at least partially because of unrealistic expectations of the tasks involved. The new core team looks like it might finally be pointing in the right direction.

Conclusions

The idea of independent free software projects is still very new. Things are changing rapidly, and it's difficult to guess what will happen in the future. A number of things have become evident, though:

- It's possible to run a small software project without significant administrative overhead. It's impossible to run a large software project without significant administrative overhead.
- A good kernel hacker is not automatically a good manager.
- The problems that large projects face do not differ significantly between volunteer and commercial projects.
- Working on a big software project has almost never been "fun". That was one of the reasons that the free operating system projects started in the first place: here was a chance to work on software free from the constraints of project management. This aspect, sadly, is a thing of the past.

In summary, I suspect that the FreeBSD project, and other similar projects, will gradually become more formalized, more like commercial operating systems. So what will distinguish them? I fear that the distinctions will become less and less as time goes on.

BSD: Past, Present and Future

Benno Rice
myinternet Ltd

Introduction / Abstract

There are several open source operating systems being developed these days that are based on the work done in the BSD releases that came from the University of California at Berkeley. Each of these projects works independantly of the others but all share code amongst themselves. This paper is a discussion of the history of the BSD projects, their differences and the strengths of each individually and of all collectively.

The Past

BSD, or the Berkeley Software Distribution, began with the Computer Systems Research Group at the University of California at Berkeley. The CSRG used the availability of the source to various versions of AT&T UNIX as a basis for various research projects and other work. The BSD projects brought a large number of advances to the UNIX system, such as virtual memory and the reference implementation of TCP/IP. Not only that, but they made the source code available to anyone who bought a copy, in the same way that AT&T (the original owners of the UNIX source code) did.

Some time after the release of 4.3BSD, the CSRG decided to release the code to their TCP/IP stack and supporting utilities independantly of the rest of the system. This was to allow third parties to use the TCP/IP code without having to acquire an AT&T source license. The license that the TCP/IP code was distributed under was incredibly liberal:

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
     This product includes software developed by the University of
     California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
```

```
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

The license has been modified subsequently. The first change was to remove the third clause which was causing problems due to people who had done subsequent work on various code wishing to be added to it. Later the fourth clause was also removed.

The BSD releases became very popular and were used as the basis of many commercial UNIX variants, such as the original SunOS and NeXT's NeXTStep. It also spawned a number of derivatives that continued on past the end of the original BSD projects. The first of these were the confusingly named 386BSD and BSD/386. BSD/386 was a proprietary commercial effort by Berkeley Systems Design, Inc. which was founded by several CSRG members, it was renamed BSD/OS as of version 2.0 and is now multiplatform. 386BSD was a more hobbyist-oriented project started by William Jolitz who made the source code freely available under the same BSD license as the original code.

The 386BSD project spawned two projects that separated from the 386BSD project due to frustration with the pace of development with 386BSD. The first was the NetBSD project, who elected to persue a more aggressively multiplatform strategy than 386BSD. The other project was begun by a number of people who initially started working on a new release of 386BSD but were halted by the withdrawal of William Jolitz's saction to use the name and trademark. These people took the 386BSD code and their patches and formed the FreeBSD project. The FreeBSD project initially concentrated on making an i386-specific user focussed distribution and has since become more multiplatform and with a focus on high-performance server environments. The OpenBSD project was started by Theo de Raadt after internal politics caused his departure from the NetBSD project. OpenBSD concentrates primarily on security and correctness.

Another BSD variant is the aforementioned NeXTStep. NeXT was a company founded by Steve Jobs after his departure from Apple. They built a series of computers that ran an OS called NeXTStep, which was based on 4.3BSD code running on a Mach microkernel. After Apple acquired NeXT, the NeXTSTEP OS was used as the basis for their next generation OS known as MacOS X and its open source component, Darwin.

The rest of this paper will concentrate on four of the open source BSD projects, namely FreeBSD, NetBSD, OpenBSD and Darwin.

The Present

Control Structures

Of the four projects, three have a fairly similar development model. FreeBSD, NetBSD and OpenBSD all have a central source repository with a group of people known as "committers" who are authorised to make changes to the repository. In FreeBSD and NetBSD, new committers are approved by a "Core Team" who are the nominal "leaders" of the projects. Changes from non-committers are submitted to committers who then review the change and apply it. OpenBSD has a similar model except that in place of the "Core Team" there is a "Benevolent Dictator" in the form of Theo de Raadt. Darwin again has a similar structure but with the primary point of control being Apple itself. Most of the committer body for Darwin is drawn from Apple employees, however there is now an OpenDarwin project that is based on Darwin and follows a structure similar to the Free- and NetBSD projects.

Compatibility

Each project is independant of the others. They all however share a common origin in the 4.3BSD and 4.4BSD releases. This gives them a level of compatibility in the areas of programming interfaces and to a lesser extent binary interfaces. It also means that at the kernel level there is a degree of similarity, although that is diminishing over time. Good examples of this are in the areas of virtual memory/memory management and device handling.

The 4.4BSD releases switched from the VM system used in previous releases to one based on the Mach microkernel. The roots of this are still apparent but much work has been done since. FreeBSD merged its swap and filesystem buffer caches, NetBSD redesigned its VM system as part of its UVM project, and Darwin uses Mach as its core and thus uses the original Mach VM system.

The device system from the original BSD releases has been replaced in all of the projects. In NetBSD and OpenBSD, the "newconfig" system is now used. In FreeBSD, a system called "newbus" is used. Both of these offer much more in the way of probing, attaching and detaching of devices while the system is running as well as the loading of new device drivers while the system is up. Darwin has a system called "I/O Kit" that is quite object-oriented and powerful.

All of these differences however are at the kernel level, and are not readily apparent to every day users of the system. Development at the user level is quite similar on all the projects. The system libraries are very similar and have near-identical APIs. The same C compiler (gcc) is used in all the projects. This provides a great ability to

move code between the projects which is very often taken advantage of. User-level and sometimes even kernel-level features and fixes very often migrate from one project to another.

An example of this is the set of USB device drivers that were developed in the NetBSD project and are now used in the FreeBSD and OpenBSD. Another is the kqueue kernel event queue facility that was developed in FreeBSD and is now available in NetBSD and OpenBSD.

The great advantage of this compatibility is that instead of providing vastly different targets for source code porting it is quite easy to port to all of them by porting to one. For example, the GNU autoconf targets for FreeBSD, NetBSD and OpenBSD are quite often almost identical. It becomes harder when some of the more esoteric features of the various projects are used, but for the most part code written for one project will compile and run on the others with little or no modification.

Real World Usage

The various projects are used in many various ways and in many various places. MacOS X is obviously used as a commercial consumer operating system on Apple's Macintosh computers. FreeBSD is widely used as a server operating system by large corporations like Yahoo! as well as getting some embedded use in systems like Juniper's router systems. NetBSD has a long history of being used as a research OS and along with OpenBSD enjoys wide use as an embedded OS due to their high level of portability.

Current Developments

All of the projects are working on new features and improving old code all the time, and each has a dedicated group of developers working on them. FreeBSD is working towards having a much more fine-grained approach to kernel locking to allow better utilisation of symmetric multi-processing machines. Both FreeBSD and NetBSD are persuing threading approaches based around the concept of scheduler activations.

Conclusion - The Future

It's easy to write off the various BSD projects as "splinters" or "forks" that end up wasting time and energy that would be better focussed on a single project. This line of thinking however ignores the benefits that the projects gain from having the other projects working alongside them. Since each project has a different set of priorities and goals, each project will solve a given problem in a potentially different way. This allows the other projects to see alternatives to their own current implementations and either replace them or incorporate elements from the other systems to enhance their own. This, at least in my opinion, gives the BSD projects a level of vitality that would be difficult to match in a unified project without having a large problem with tracking competing implementations of either new code or enhancements to old code. Having many independant projects means that a number of implementations can be developed and tested without disrupting the project itself.

Furthermore, the open nature of the projects in terms of their source code gives a lot of freedom to end users of the software. While this means that it can be a lot harder to ascertain whether a given vendor is using BSD code in their products, the vendors themselves can use the software freely and easily. All in all this points to the BSD projects or their progeny being around for a long time to come.

What You See is Not Always What You Sign

Audun Jøsang

Distributed Systems Technology Centre Level 12, S-Block, Queensland University of Technology, GPO Box 2434, Brisbane Qld 4001, Australia
tel:+61-7-3864 5120, fax:+61-7-3864 1282
email: ajosang@dstc.edu.au

Dean Povey

Wedgetail Communications
Level 14, 388 Queen Street, Brisbane Qld 4001, Australia
tel:+61-7-3023 5100, fax:+61-7-3023 5199
email: povey@wedgetail.com

Anthony Ho

Distributed Systems Technology Centre
Level 12, S-Block, Queensland University of Technology, GPO Box 2434, Brisbane Qld 4001, Australia
tel:+61-7-3864 5120, fax:+61-7-3864 1282
email: anthonyh@dstc.edu.au

Introduction

The concept of "digital signature", first publicly described by Diffie and Hellman (1976) [5] in their classic paper "New directions in Cryptography", suggests that it is a computer-based equivalent of physical written signatures. Although there are similarities between handwritten and digital signatures there are also fundamental differences. The main similarity is that both types of signatures can provide evidence of authenticity of a document. The differences are due to the radically different nature of paper based documents on the one hand and digital documents on the other. In paper-based transactions a document consists of text printed as ink on a piece of paper, where the text represents the information and the paper represents the storage medium. In this way the information and the storage medium are inseparable. The validity of a paper-based document is authenticated by a signature written in ink on the same piece of paper. The signature serves as evidence of the signer's agreement to the text on the paper, and the only instruments between the signer and the document are optional glasses for better viewing the text and a pen for applying the signature, both of which can be considered reliable. For digital signatures all of this changes. Documents are immaterial because the information is represented by logical bits that can be stored on, and copied to, any suitable electronic medium, and they only become meaningful to humans when represented through an analogue physical medium such as a computer screen or a printout. The validity of a digital document is authenticated by verifying that an immaterial digital signature logically matches the already immaterial document. Because a digital document in its immaterial form can not be observed directly by the signer the digital signature can only serve as evidence of the signer's agreement to some analogue representation of the document although it is usually assumed that it represents the signer's agreement to the immaterial document itself. Highly complex instruments are now needed not only for viewing the document but also for producing the digital signature. Assessing the consistency and reliability of these instruments is beyond the reach of most signers.

The computer platform is the most complex element in the chain between a digital document and its analogue representation. We will not discuss platform integrity in detail, but simply mention that it is a fundamental problem, not only for correctly generating and verifying digital signatures, but for IT security in general. If the platform is

not secure, then no application running on top of it can be made secure. See e.g. Loscocco *et al.* (1998) [11] for a general discussion of this topic. A more specific description of attacks on digital signatures through attacking the platform integrity can be found in Cremers *et al.* (2001) [4]. A description of some integrity vulnerabilities in the framework for digital certificates and authentication on the Web is described in Jøsang *al.* (2001) [10] and Redhead & Povey (1998) [15]. A discussion about the binding between digital documents and digital signatures can be found in McCullagh et al. (2001) [13].

This paper focuses on principles for encoding and representing digital documents, such as XML, HTML, ASN.1 and font type specifications. The flexibility that is deliberately built into these standards can make the representation and interpretation of the documents inconsistent and unreliable. When digital documents are being signed it can therefore be difficult to determine the exact content to which the digital signature applies.

Digital Signatures on XML Documents

False Positives and False Negatives in XML

The authors of the draft XML Signature Syntax Processing specification (Bartel *et al.* 2001) [2] are aware of potential problems when digitally signing XML formatted documents. Not only can the same XML document look different in two different applications, but different XML documents can look the same in the same application. When an XML digital signature is applied it can therefore be difficult to know exactly what is being signed. A distinction can be made between syntactic form and the semantic contents of XML documents. An important question to be answered in this context is what determines whether two different XML documents can be considered semantically equivalent, depending on their syntactic form and context.

The first case to consider is when two XML documents are semantically equivalent despite having different syntactic form. This can lead to false negatives, i.e. meaning that two documents that are semantically equivalent are not recognised as such by a human or an application. This is possible because humans are used to thinking that different form implies different meaning, and applications are designed to only make distinctions based on form. The use of canonicalisation can assist in avoiding false negatives.

Canonicalisation (Boyer 2001) [3] is a transform whereby XML documents are transformed into a canonical syntactic form. The idea is that two XML documents having the same canonical form can be considered semantically equivalent within a given application context. However, canonicalisation does not mean that two documents are semantically equivalent *if and only if* their canonical form is identical ie. there can be cases within particular applications where two documents are semantically equivalent even though they have different canonical forms.

The second case to consider is when two XML documents are semantically different despite having the same syntactic or canonical form. This can lead to false positives, i.e. when two documents are wrongfully seen as semantically equivalent by a human or an application. This is possible because the meaning of an XML document can depend on external element type definitions (i.e. the schema, DTD, or natural language description) that are associated with the XML document, but are not part of the signature. If any of these change, the meaning of the same XML document can also change. For example, changing the DTD of an XML document does not effect its canonical form, so in the case of XML signatures, the digests will still match and the signature will verify, but the document could now have an entirely different meaning.

When an XML document is digitally signed, the signer's consent expressed by the signature applies to the semantic meaning of the document, whereas the digital signature mechanisms itself is an automatic process that ignores the meaning and only applies to the syntactic form. A distinction can thus be made between what the signer believes is being signed and what is physically being signed. Whenever false positives and negatives can occur, digital signatures are vulnerable to attack. The example below illustrates how a case of false positive can be exploited for fraudulent purposes.

Example A: Same XML but Different Meaning

This example illustrates how modifying the `ATTLIST` can change the meaning of an XML document. In the example, the author of the sample XML is a disgruntled screenwriter, and for the next episode of the `http://www.etsi.org/getastandard/home.htm` soap that he is working on he produces a plot outline like this:

```
<!DOCTYPE Soap [
  <!ELEMENT Character ... >
  <!ATTLIST Character
      name     ID    #REQUIRED
      nickname CDATA #IMPLIED
  >
]>

<Soap>
```

```
  <Characters>
    <Character name="alice"> ... </Character>
    <Character name="bob">    ... </Character>
    ...
    <Character name="susan" nickname="alice"> ... </Character>
    <Character name="tim"   nickname="bob"  > ... </Character>
  </Characters>
  <Actions>
    <Kill killer="#alice" victim="#bob" />
    <!-- (Uses URI fragments but could also use IDREF attributes) -->
  </Actions>
</Soap>
```

and gets sign-off from the series producer. (For concreteness, say he signs the entire document using an enveloped signature, and puts the signature just before the closing "`</Soap>`"). Now the author modifies the `ATTLIST` declaration, changing the `ID` attribute from "name" to "`nickname`", and end up with

```
<!DOCTYPE Soap [
  <!ELEMENT Character ... >
  <!ATTLIST Character
      name     CDATA #REQUIRED
      nickname ID    #IMPLIED
  >
]>

<Soap>
  <Characters>
    <Character name="alice"> ... </Character>
    <Character name="bob">    ... </Character>
    ...
    <Character name="susan" nickname="alice"> ... </Character>
    <Character name="tim"   nickname="bob"  > ... </Character>
  </Characters>
  <Actions>
    <Kill killer="#alice" victim="#bob" />
    <!-- (Uses URI fragments but could also use IDREF attributes) -->
  </Actions>
  <Signature xmlns="..."> ... </Signature>
</Soap>
```

This still verifies, but now Tim dies instead of Bob, which isn't the outcome that the series producer thought he was signing.

Avoiding False Positives

Digital signatures on XML documents will be of limited value if the type of vulnerabilities illustrated in the previous example can not be avoided. In Bartel *et al.* (2001) [2] it is recommended that XML documents are canonicalised prior to digital signing, and that element type definition be signed together with the document.

Canonicalisation prior to digital signing is a transform mechanism with the purpose of avoiding false negatives. Including element type definitions in the digital signature would be a mechanism to avoid false positives, however this is not a requirement for XML signatures, therefore making them (in a minimal required state) vulnerable to the type of attack illustrated in Sec. 1.

These definitions could be included in the signature by either adding a reference to an external DTD or a copy of an internal DTD before signing. Defining the element types using XML schema (as opposed to DTD) is even more effective, since the ability to force a 'malicious' DTD upon a recipient is not as trivial when using schema, due to the way schemas are referenced.

Only by requiring identical element type definitions in addition to canonical form can this equivalent semantical contents be guaranteed.

Digital Signatures on ASN.1 Documents

False Positives and False Negatives in ASN.1

ASN.1 documents have a particular syntactic form defined by the ASN.1 standard [7]. Associated with every ASN.1 document is also a semantic content which is determined by how the document is interpreted by humans or applications. In this context it is important to find out what determines whether two different ASN.1 encoded documents can be considered semantically equivalent, depending on their syntactic form and context. This gives an indication

of whether false positives and false negatives can occur, and thus whether digital signatures on ASN.1 documents are vulnerable to attacks.

The first case to consider is whether two ASN.1 documents can be semantically different despite having the same syntactic form. This can lead to false positives, i.e. when two documents are wrongfully seen as semantically equivalent by a human or an application. ASN.1 documents do not depend on external definitions in the same way XML documents do. For XML, external elements are explicit, whereas for ASN.1 they are implicit by assuming that everyone share a common understanding of the encoding. Under that assumption, two ASN.1 documents that are syntactically equal are thus guaranteed to be interpreted equivalently by humans or applications, thus eliminating the possibility of false positives.

The second case to consider is whether two XML documents can be semantically equivalent despite having different syntactic form. This can lead to false negatives, i.e. meaning that two documents that are semantically equivalent are not recognised as such by a human or an application. This is possible because humans are used to think that different form carries different meaning, and applications are designed to only make distinctions based on form.

There are various encoding rules for encoding the same (semantic) information, (i.e. BER, CER, DER [8] and PER [9]), opening the possibility of false negatives. The following example describes how this vulnerability can be exploited.

Example B: Failed Revocation of Digital Certificates

A particular security vulnerability described by Manger 2001 [12] exists in the 3GPP (Third Generation) Mobile Execution Environment (MExE) specification [6] relating to certificate revocation lists (CRLs). A detailed analysis of the vulnerability and a suggested solution has been sent to various parties involved with the MExE specification development (at T-Mobile, Lucent, Motorola & Vodafone).

MExE defines a certificate configuration message (CCM) that can act like a certificate revocation list (CRL) – listing certificates that must be disabled. The CCM CRL identifies a certificate by its "fingerprint" – a hash of the complete certificate. [Note: this quantity is called a "thumbprint" in Microsoft's certificate tools.] It is possible (and easy), however, to modify the encoding of the unsigned portions of a certificate to change the fingerprint without invalidating the signature. For instance, use indefinite-length BER for the outer wrapper, instead of DER. If the CRL issuer and relying party (MExE device) have different encoding of the same certificate the fingerprint in the CRL will not match the fingerprint calculated by the MExE device so the certificate will not be disabled when it should be.

Manger 2001 [12] describes a number of different encodings of a single certificate and each has a different fingerprint. All are (legitimately) accepted by basic ASN.1 runtime libraries. Most are (legitimately) verified as correct by Windows NT certificate validation software.

Avoiding False Negatives in ASN.1

The vulnerability in [6] exists because the hash of a received certificate can be taken of **any** particular encoding such as DER or BER, rather than of a **canonical** encoding such as CER and to a certain extent DER. When the specification allows hashing of non-canonical encodings, it is evident that hash values may not match. One possible solution is to only hash what is being signed. Another solution is to always compare hash values of ASN.1 documents based on canonical encoding such as DER and CER. This will effectively eliminate the type of false negatives describe in the above example.

Manipulation of Font Types

The Role of Font Types for Correctly Viewing Digital Documents

The specification of how the information in a document, encoded as immaterial bits, is displayed as an analogue image on a computer screen or a printout, is crucial for correctly bringing the meaning of a document to the attention of a human observer, for example prior to applying a digital signature. Central to this discussion lies the distinction between characters and glyphs. Characters are codes assigned by the Unicode standard [16] which represent the smallest semantic units of language, such as letters. Glyphs are the specific forms that those characters can take, as defined by the font type. One character may correspond to several glyphs providing alternate forms of the same letter within the same font type. This can be achieved by additional layout features of a given typesetting standard such as e.g. OpenType [1]. One glyph can also represent multiple characters, as in the case of the "ffi" ligature, which corresponds to a sequence of three characters: f, f and i.

Although a character is usually associated with some graphical form, it is not up to the Unicode standard to dictate that form in every detail, the standard only indicates one typical form for each character. Theoretically it would be

possible to define characters as sounds instead of graphical forms, in the same way as it is possible to play sequences of characters as audible words instead of displaying them as written words. The actual shape of a character in a particular application is defined by the specification of the font type used by the application and the font interpreter for the particular analogue device, such as a computer screen or a printer. Ultimately, the interpreter generates a pixel matrix for each character as a function of the shape and size of the glyph and the resolution characteristics of the analogue output device. The correspondence between the bits that constitute a character code and the analogue display of a glyph is conceptually illustrated on Figure 1 below.

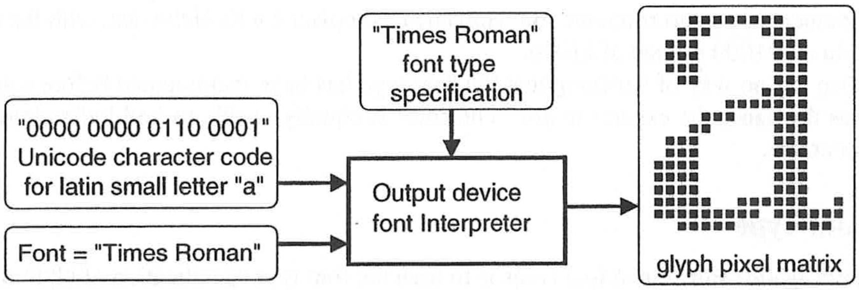


Figure 1: Correspondence between character code and glyph

Depending on the document type and application, digital documents can be defined with or without specifying the font. In case the font is not specified, the application will use some default system font when displaying or printing the document. In case the font is specified, the application will try to use that font when displaying or printing, but if it is unavailable on the system, will substitute the font with a default or a similar font. When substitution takes place, it can happen tacitly or with a warning to the user, depending on the application and configuration. In the standard configuration of Microsoft Word for example, no warning is given. By looking at the formatting toolbar however, it is normally possible to find out what font type was originally applied to a character or a block of text. But the formatting toolbar does not necessarily give reliable evidence because utilities exist for MSWindows (e.g. *TrueType Font Namer* [14]) to change font type names, so that in practice any font type can have any name. Although there are a limited number of commonly used font types there can in theory be an arbitrary number of different font types. Within the same MSWord document for example, each character can have a different font type.

The above discussion indicates that the font name and the font type specification are variables that can change the optical appearance of digital documents. This flexibility can of course also be misused with malicious intent, in particular in case of digital signatures. We will describe two possible scenarios for how this can happen.

Example C: Substituted Fonts

In the first scenario, Clark prepares a digital document with the following contents displayed in Helvetica font:

On 24 October 2001, Alice borrowed from Clark the sum of $1000.

However, Clark defines a new font called "Helvetika" which is similar to Helvetica in style, but for which the glyphs for "`A`", "`i`", "`c`" and "`e`" have been interchanged with the glyphs for "`C`", "`a`", "`r`" and "`k`", and installs this font on his computer. Clark then applies this new font to the words "`Alice`" and "`Clark`" with the result that when viewed on his computer the document looks like:

On 24 October 2001, Clark borrowed from Alice the sum of $1000.

Clark then borrows $1000 from Alice and digitally signs the document. Alice is satisfied by visual inspection of the document and verification of the digital signature on Clark's computer. Alice copies the digitally signed document to her floppy disk as evidence for Clark's debt to her. When Alice tries to prove her case and displays the digitally signed document on a court room computer the font Helvetika is replaced with Helvetica or some other default font, with the result that the evidence becomes valueless.

In this scenario, it would have been possible for Alice to detect that there was something wrong had she checked the fonts used in the document before signing. For a small document like in this example that might be feasible, but it soon becomes impractical when the size increases to more than a few lines. The court can also interpret the unknown font type Helvetika as evidence that the document could have been manipulated with malicious intent.

Example D: Changed Font Names

In the previous example it was theoretically possible to detect the fraud. This example shows how it is possible to avoid detection by hiding the fact that font substitution takes place.

In this scenario, Clark prepares a digital document with the following contents when displayed in Helvetica font:

On 24 October 2001, Clark borrowed from Alice the sum of ¥1000.

Clark then creates a font type which is similar to Helvetica in style, but for which the glyphs for "¥" and "$" are interchanged. Clark calls this new font type "Helvetica", but we will call it "Helvetica′" in order to distinguish it from the original one. Clark substitutes the original Helvetica font type with Helvetica′ on his computer, with the result that the document looks like:

On 24 October 2001, Clark borrowed from Alice the sum of $1000.

Clark then borrows $1000 from Alice and digitally signs the document. Alice is satisfied by visual inspection of the document and verification of the digital signature on Clark's computer. Alice copies the digitally signed document to her floppy disk as evidence for Clark's debt to her. When Alice tries to prove her case and displays the digitally signed document in a court room the font Helvetica′ is replaced with Helvetica, with the result that the evidence indicates a debt of ¥1000 instead of $1000.

In this scenario Alice has no way of verifying that the font type has been manipulated before signing, because Clark's new font carries the name she expects to see. The court is equally unable to find indications of malicious manipulation of the document.

Controlling the Font Type

A simple countermeasure against substituted font types is to hash the font type specification of all font types used in the document to be signed, and to include the hash in the digital signature, and to disallow substitution of font types. By caching the hash value of each font type specification performance deterioration will be minimal. Disallowing font type specification will only cause a minimal reduction in the flexibility of document exchange. In practice, it will mean that exotic and customised font types in digitally signed documents should be avoided because they are likely to make signature verification impossible.

Digital Signatures on Graphical Features

Digital documents can contain more than just text. Common examples of additional graphical features are pictures, drawings, table formatting and background colour. The correct and consistent rendering of such features is crucial for the meaning of digital documents. If graphical features are displayed differently in different applications, then the meaning is likely to change, making it meaningless to digitally sign such documents. The example below shows how inconsistent handling of table tags in HTML can make the same table look completely different.

Example E: Inconsistent Handling of HTML Table Tags

In this scenario it is assumed that three building contractors have submitted a quote for building an office building for a company. The quotes are: "Alice Architects and Builders": $800,000, "Bob Building Contractors":$900,000, and Clark Constructions": $1,000,000. The company managers who evaluates the quotes is satisfied with the qualifications of all three contractors, and decides to list his preference as a function of price. The manager asks the company's web editor to create a table with the building contractors listed according to price. The manager digitally signs the html page and submits it to the company's procurement department for further processing.

What the company manager does not know is that the web editor is a close friend of Clark, and therefore will try to make "Clark Constructions" win the contract. The web editor knows that the manager uses Microsoft Internet Explorer, whereas the procurement department uses Netscape Navigator. The Web editor encodes the HTML table so that "Alice Architects and Builders" gets highest preference when viewed with Microsoft Internet Explorer, and "Clark Constructions" gets highest preference when viewed with Netscape Navigator. The figures below show what the tables look like when viewed in Microsoft Internet Explorer and Netscape Navigator respectively.

Figure 2 shows what the managers sees and signs. Figure 3 shows what the procurement department sees. Note that that the HTML code is identical in both cases.

This is possible because the HTML tag `tfoot` is handled inconsistently. Microsoft Internet Explorer always creates a row at the end of the table whereas Netscape Navigator creates a row at the point where the tag appears in the HTML code. Thus by encoding "Clark Constructions" with the tag `tfoot` just after the table head, that row will appear to be the last table entry in Microsoft Internet Explorer, and the first table entry in Netscape Navigator. The complete HTML code for this example is shown below.

```
<!DOCTYPE html>
<html>
<head>
<title>Table example showing difference between
       IE and NN</title>
</head>
```

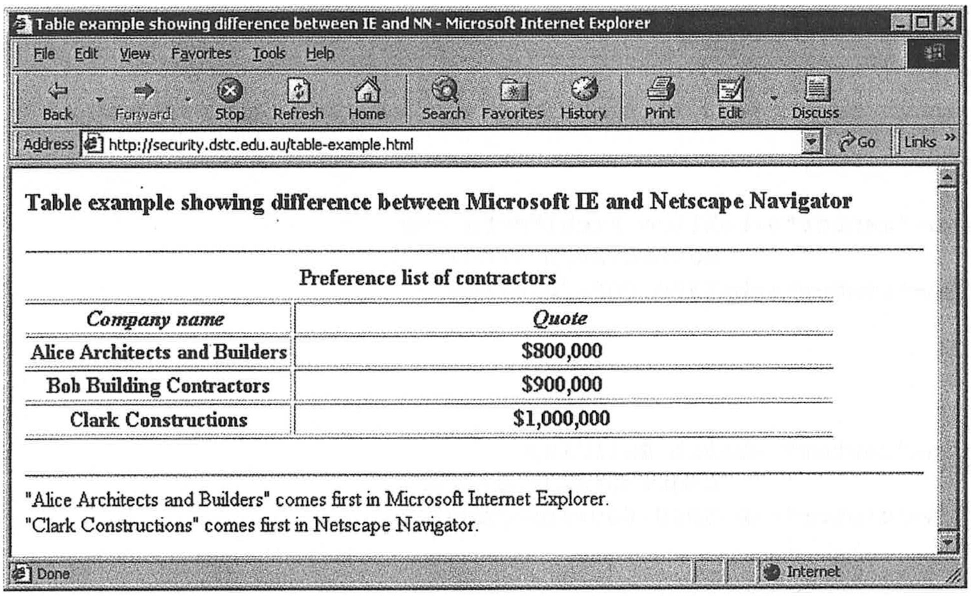


Figure 2: Viewing the table in Microsoft Internet Explorer

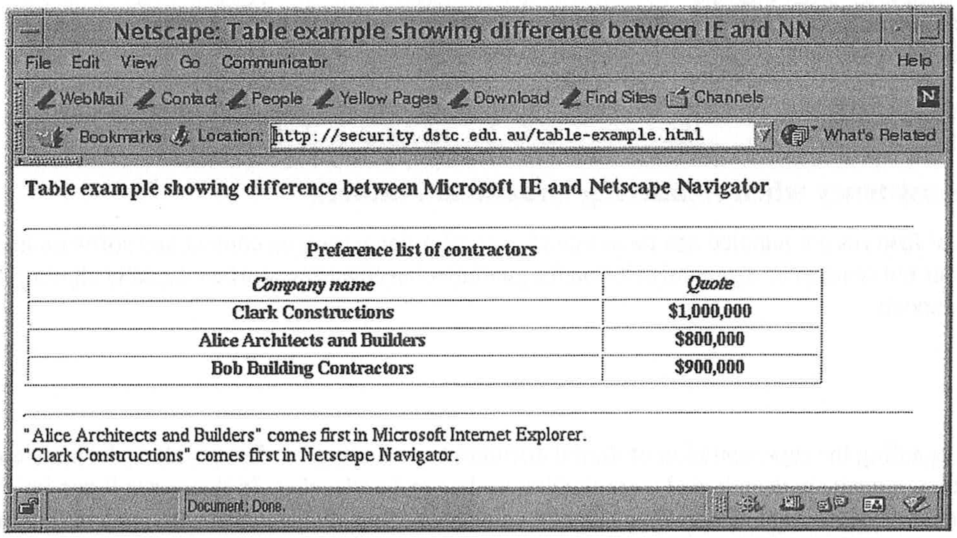


Figure 3: Viewing the table in Netscape Navigator

```
<body bgcolor="#ffffff">
<h3>Table example showing difference between
Microsoft IE and Netscape Navigator</h3>
<hr />
<table border="1" width="90%" align="centre"
       frame="hsides" rules="groups">
  <caption><b>Preference list of contractors</b></caption>
<colgroup width="20%" align="center" valign="top">
<colgroup span="2" width="40%" valign="top">
<thead>
  <tr>
    <th><i>Company name</i></th>
    <th><i>Quote</i></th>
  </tr>
</thead>
<tfoot>
  <tr>
```

```
    <th>Clark Constructions</th>
    <th>$1,000,000</th>
  </tr>
</tfoot>
<tbody>
  <tr>
    <td align="center"><b>Alice Architects and
                          Builders</b></td>
    <td align="center"><b>$800,000</b></td>
  </tr>
</tbody>
<tbody>
  <tr>
    <td align="center"><b>Bob Building
                          Contractors</b></td>
    <td align="center"><b>$900,000</b></td>
  </tr>
</tbody>
</table>
<br>
<hr />
"Alice Architects and Builders" comes first in
Microsoft Internet Explorer.<br>
"Clark Constructions" comes first in Netscape Navigator.
</body>
</html>
```

Avoiding Inconsistency when Rendering Graphical Features

The way graphical features are handled can be extremely complex depending on context and software implementation, and there does not seem to be a general solution for guaranteeing consistency when digitally signing documents with graphical elements.

Discussion

Inconsistencies regarding the representation of digital documents normally pose few problems because authors and readers usually have a positive attitude and a goodwill to understand each other. In that sense it can be argued that most digital documents can be signed with little risk of misunderstanding. Unfortunately this way of reasoning is not viable for security. Designing secure solutions requires the opposite way of thinking, and the question to be asked is: *What if somebody deliberately tries to cause misunderstanding, would he or she succeed?* The examples described in this paper shows that it is relatively easy to create confusion regarding the semantic content of digital documents and thus that it can be difficult to know what a digital signature applies to.

Section 1 and Section 1 mention the use of canonical form for uniquely representing digital documents in XML and ASN.1. As mentioned in Section 1 however, canonicalisation does not guarantee that semantically equivalent documents necessarily must have identical canonical form. It is extremely difficult to completely avoid ambiguity in the specifications of canonical forms, making it almost impossible for software vendors to write implementations that produce exactly the same results. Canonical form can therefore only give an indication, and not a guarantee, that two documents are semantically equivalent.

Some of the examples described above had the purpose of creating specific alternative meanings to digital documents. Another form of attack could be to simply create ambiguity about a document's meaning, and thereby make the document unacceptable or inadmissible as evidence. An attacker could for example hide ambiguity in a digital contract in order to have the freedom at a later stage to claim that the contract is invalid on that basis and thereby be liberated from the contractual obligations.

A simple conclusion to be drawn from our analysis is that the more flexible and open the framework for handling digital documents is, the more complex the correspondence between the digital form on the one hand and it's interpretation or analogue representation on the other becomes. It is in this space that the vulnerabilities we have described are found and that attacks can be mounted.

A distinction can be made between machine interpretation and human interpretation of documents. The first category applies to XML and ASN.1 documents that can be interpreted, digitally signed and verifued by software and hardware without direct human intervention. In this case it should in principle be possible to completely avoid

false positives and false negatives. The failure of XML Digital Signature and the MExE specifications to achieve this can only be attributed to specification flaws that can and should be rectified.

In the second category where human interpretation of digital documents depends on visual inspection, ambiguity can be avoided by signing the bitmap that constitutes the analogue graphical representation of a digital document. This applies to all examples mentioned in the previous sections, including XML and ASN.1 documents. As mentioned in Bartel *et.al* (2001) [2] this would result in data that are difficult for software applications to subsequently manipulate. A possible solution is for the signer to always archive an analogue image of all documents he or she signs, and to include the hash of that image in the digital signature. In that way it is possible to refer to what the signer really saw at the moment the signature was applied, should it be required at a later stage.

Conclusion

The term "digital signature" is a metaphor that can make people falsely believe that it is equivalent to handwritten signatures. However, it should be seen as a new paradigm proper to computer systems rather than treating it equivalent to handwritten signatures. The original form and analogue representation are completely separate phenomena for digital documents, whereas paper based documents have both combined into one. Whenever the meaning of a digital document can only be accessed by viewing an analogue representation of it, the consent expressed by the digital signature should only apply to what the signer sees, i.e. to its analogue representation, and not to the immaterial bits that constitute the digital document in its original form.

Acknowledgements

Thanks to Thomas Maslen from Wedgetail Communications for providing the XML code for the soap plot in Section 1. Thanks to Andrew Mich from Telstra Research Laboratories for providing the example about ASN.1 in Section 1.

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

Bibliography

[1] Adobe. *OpenType User Guide*. Adobe, 2000. URL: http://www.adobe.com/type/browser/pdfs/OTGuide.pdf (visited 18 September 2001).

[2] Mark Bartel et al. *XML-Signature Syntax and Processing - W3C Proposed Recommendation 20 August 2001*. W3C (World Wide Web Consortium), 2001. URL: http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/.

[3] John Boyer. *Canonical XML, Version 1.0 - W3C Recommendation 15-March-2001*. W3C (World Wide Web Consortium), 2001. URL: http://www.w3.org/TR/2001/REC-xml-c14n-20010315.

[4] Armin B. Cremers, Adrian Spalka, and Hanno Langweg. The fairy tale of 'What You See Is What You Sign - Trojan Horse Attacks on Software for Digital Signatures. In *IFIP Working Conference on Security and Control of IT in Society-II (SCITS-II)*, Bratislava, Slovakia, June 2001.

[5] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[6] ETSI. *ETSI TS 123 057 V4.2.0 (3GPP TS 23.057 V4.2.0), UMTS Mobile Execution Environment (MExE) Functional Description, Stage 2*. European Telecommunications Standards Institute, June 2001. URL: http://www.etsi.org/getastandard/home.htm (visited 01.10.2001).

[7] ISO. *IS 8824-1,2,3,4. Abstract Syntax Notation One (ASN.1). Part 1: Semantic model, Part 2: Information object specification, Part 3: Constraint specification , Part 4: Parameterization of ASN.1 specifications*. International Organisation for Standardization, 1998.

[8] ISO. *IS 8825-1. ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. International Organisation for Standardization, 1998.

[9] ISO. *IS 8825-2. ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)*. International Organisation for Standardization, 1998.

[10] A. Jøsang, P.M. Møllerud, and E. Cheung. Web Security: The Emperors New Armour. In *Proceedings of the European Conference on Information Systems (ECIS2001)*, Bled, Slovenia, June 2001.

[11] Peter A. Loscocco et al. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *Proceedings of the 21st National Information System Security Conference*, pages 303–314. NSA, October 1998.

[12] James Manger. Vulnerability in 3GPP (Third Generation) Mobile Execution Environment (MExE) specification relating to certificate revocation lists (CRLs). Defect report to 3GPP standartisation working group, 2001.

[13] Adrian McCullagh, William Caelli, and Peter Little. Signature Stripping, A Digital Dilemma. *Journal of Information, Law and Technology*, 6(1), 2001.

[14] UniTech / MyTools. TrueType Font Namer. URL: http://www.mytools.com/fontnamer.html (visited 20 September 2001.

[15] Tim Redhead and Dean Povey. The Problems With Secure On-line Banking. In *Proceedings of the XVIIth annual South East Asia Regional Confereance (SEARCC'98)*, July 1998.

[16] UNICODE. *The Unicode Standard, Version 3.0.* Addison Wesley Longman Publisher, isbn 0-201-61633-5 edition, 2000. URL: http://www.unicode.org/unicode/standard/standard.html (visited 20 September 2001).

Linking Chains - A methodology for developing rules for IP Chains

Daniel Bradley
Eric Faccer
Mark Cross

Distributed Systems Technology Centre
Level 12, S Block, QUT Gardens Point
Brisbane Qld 4001, Australia

Introduction

Due to the large number of vulnerabilities in today's applications and services, network firewalls have become a common security measure. However, if configured incorrectly, what should be a security measure becomes a security liability.

While there are sources of information that describe the designs and architectures of firewalls, few sources describe how to build the rule sets that are critical for firewalls to meet their intended purpose. This paper intends to address this imbalance by describing a methodology that can be used to formulate rules for the IP Chains packet filtering firewall [1].

The remainder of this paper is organised as follows; section two provides an introduction to firewalls, explaining their purpose and detailing the security vulnerabilities that need to be taken into consideration during configuration. Section three gives an overview of the IP Chains packet filter, describing how Linux handles incoming packets, and how IP Chains decides their fate. Section four presents our methodology by producing an IP Chains configuration script that enforces the network traffic policy of an example network. Section five outlines future work to be done.

Firewalls

The term firewall describes a safety measure that stops fire in one area spreading to another area. In the building industry it is a protective layer that stops a fire from spreading from one apartment to another; in the automotive industry it is the barrier that shields the passenger compartment from the engine. To generalize the purpose of a firewall is to stop problems from spreading.

A network firewall is a system of components designed to control the flow of network traffic between networks. This system is usually composed of *packet filters*, which make decisions based on the contents of the packet header, and *application proxies*, which act as middlemen between clients and services. In some instances these components are present on the same machine, but more usually they are located on an adjacent network, commonly referred to as a DMZ (demilitarised zone).

The difference between *packet filters* and *application proxies* can best be explained by identifying the different layers of the TCP/IP stack. Figure 1 shows the TCP/IP stack compared against the OSI model [2]. Each layer encapsulates the previous layer, adding layer relevant information as a header.

Application proxies work at the application layer and typically act as a man-in-the-middle, terminating the connection to the client and establishing another to the server. When the proxy server receives a request from a client, it may check its validity before making the request to the server on the client's behalf. No actual IP packets are passed between the client and server. The trade-off is that application proxies are typically limited to one particular protocol, e.g. http. An example of an application proxy system is the "TIS Firewall Toolkit" [3].

Packet filters work at the transport layer, either dropping or forwarding packets, by inspecting their network and transport headers.

Configuring a packet filter is a non-trivial task. Below we discuss some non-obvious aspects of TCP/IP communication that need to be checked by all packet filters.

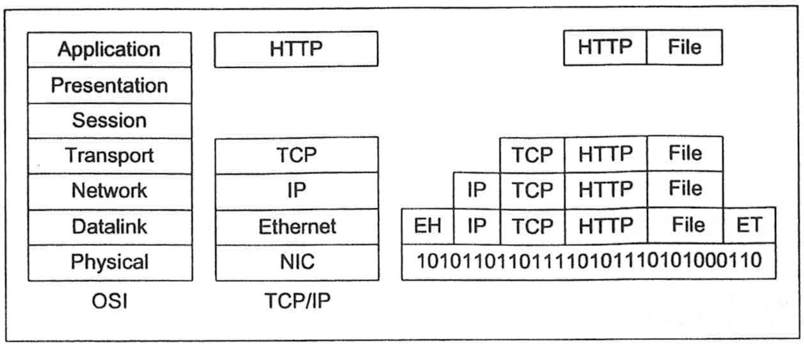


Figure 1: Network layers

What a packet filter should filter

When the IPv4 family of protocols were developed security was not an issue, however, subsequent widespread deployment of these protocols has created an environment in which security is an issue. Below we discuss several security weaknesses that need to be addressed when configuring a firewall.

Malformed packets

TCP/IP implementations were designed to expect packets that meet the published TCP/IP specification. Malformed packets do not. When a host can only process specification compliant packets, and it receives a non-conformant packet, the behaviour is undefined.

Two common ways for packets to be malformed are for the TCP flags to be set incorrectly, e.g., two of the SYN, ACK, or FIN, flags set at once; and for the packet length to be different from the actual length. A description of the different ways, intentional and unintentional, that packets have been malformed can be found in [4].

An example of an attack using malformed packets is the attack known as the "Ping of Death" [5], which sends an ICMP *echo* request message that is larger than expected. This causes the packet buffer to overflow, usually causing the machine to crash [6].

The IPChains system does not provide a way to test whether packets are malformed in any way, and must rely upon a sanity check that is performed when packets are received by the system.

Spoofed source traffic

Many applications use a packet's source address for some degree of authentication, e.g. NFS, and TCP wrappers. By spoofing the address (planting another address in the packet's source address field) it is possible to abuse these trust relationships.

Non-connection oriented protocols such as ICMP and UDP are more susceptible to these attacks due to their lack of a handshake protocol, which requires communication in both directions before a connection is established. An example is the *chargen* attack, where a loop is created between a *chargen* service and an *echo* service, producing a packet storm [7].

A problem with trying to establish a TCP connection using a spoofed source address is that the reply goes to the spoofed host, not to the actual source. Therefore to receive the reply packet it is necessary for the attacker to either; guess the reply, sniff the packet from the network, or perform some form of RARP poisoning [8]. Despite these difficulties, attacks have been successfully carried out using a spoofed TCP connection [9].

It is recommended that at the very least packet filters drop spoofed traffic by performing *ingress* filtering, i.e., dropping any packets that arrive on an interface they shouldn't. This protects the network from external attacks using spoofed packets and from being used as a base for attacks [10].

The *nmap* [11] utility may be used to determine if your routers will pass spoofed packets. The following command uses the –S argument that spoofs the source address on packets sent.

```
nmap <target IP> -S <source IP> -e eth0 -r -sS
```

A useful feature, when wanting to intercept a reply packet to a spoofed address, is the Internet protocol's *source routing* option. This allows the sender to dictate the route that a reply packet travels over the Internet, allowing them to route the packet through a network they control; where it can be conveniently sniffed. Therefore, it is recommended that source routing be disabled; although this may interfere with some mobile IP schemes.

Broadcast traffic

Broadcast addresses are used when a host wants to send a packet to every host on a particular network. This is usually when a host does not know who to contact for some service, or is trying to broadcast information about itself. A common use of broadcasting is the DHCP (Dynamic Host Configuration Protocol) [12], which sends a packet to the broadcast address and receives a reply in a similar fashion. Directed broadcasts are broadcasts targeted at a particular network.

The misuse of broadcast traffic has been used in attacks such as the Smurf attack [13]. In this attack a series of spoofed ICMP *echo* request packets are sent to a network's broadcast address. If the router allows *directed broadcasts*, these requests will be forwarded on to each host on that network. The result of this is that the host at the address that was spoofed can be deluged with ICMP echo response packets.

Another related issue is that if broadcast packets are not specifically denied or allowed, they tend to be logged; this may obscure more important log messages.

Improper traffic

The Internet protocol address range is broken up into a number of different classes. Only four of these are intended to be routable across the public Internet, one of which is only used for multicast traffic. The others are reserved for future use.

In general an Internet router should not route any traffic containing addresses belonging to the *private* address ranges [14], or any addresses that match the address mask "240.0.0.0/5", which are reserved and undefined addresses.

ICMP and IGMP packets

ICMP (Internet Control and Management Protocol) and IGMP (Internet Group Management Protocol) packets are used for sending control and management information about connections. They can be used to perpetrate denial of service attacks such as the ping of death [15]. ICMP can also be abused to allow scanning of a network [16]. Possible information that can be acquired from such a scan includes remote operating system type, remote network topology, and the existence of a remote host, this is called enumeration.

ICMP messages can be grouped into two classes: error messages, and query messages. Many of the query messages are now deemed obsolete by specification and should be blocked. For example, *information request & information reply* (types 15,16) messages.

Of the error messages, the *destination unreachable* messages (type 3) are the most important for network performance. These prevent lengthy timeouts when a host is unable to respond [17]. These messages are also required for path MTU discovery, a process important for optimising bandwidth usage and preventing network degradation.

Another error message, *source quench* messages (type 4) are required for transport layer flow control. Other ICMP messages can be blocked safely without adversely affecting network performance, but doing so will lead to loss of certain types of functionality. For example, the ***ping*** tool relies on being able to send *echo* messages (type 8) and receive *echo reply* messages (type 0). The ***traceroute*** tool uses *time exceeded* (type 11) messages to view a datagram's route through a network. This gives information about the network's logical topology and is useful for detecting routing loops. This information, useful for a system administrator is also potentially useful for an attacker building a detailed view of a network, and as such should be carefully considered in a network security policy.

However, in certain cases, allowing certain messages (types 8,0,11) may be deemed acceptable due to mitigating factors. For example, it could be considered futile to block *echo* messages to prevent enumeration of a networked host if that server is vulnerable to higher protocol layer scan technique such as TCP port SYN scanning. Thus, blocking ICMP messages to prevent network enumeration should be reserved for workstations that do not listen on ports.

Local router traffic

Due to the crucial role of the packet filter - enforcing an organisation's network traffic policy - it is necessary that it be heavily protected. This is achieved by ensuring that there are few, if any, services running on the machine, that only those services are allowed to accept packets, and only certain machines are allowed to connect to them.

In most cases there is only a single service for remote administration. With Linux based firewalls this is usually the *secure shell* daemon. It is easy, however, to make simple mistakes that could allow external access to this service. In particular an assumption that a person might make is that if a service is bound to the internal interfaces' address then only internal machines will be able to connect to the service. This is not true, as, by default some routers will route to any of its interfaces [18].

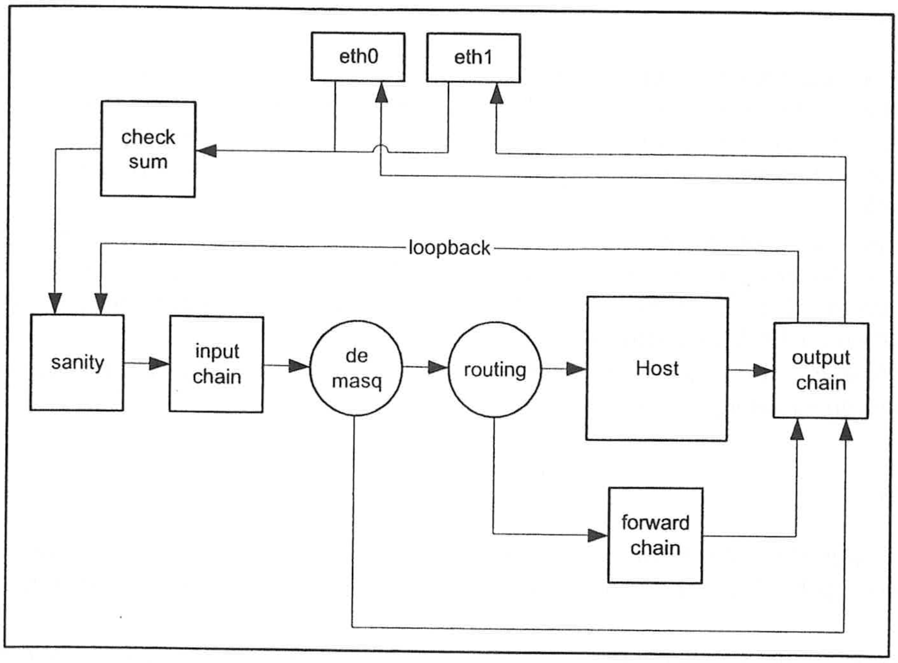


Figure 2: How packets traverse the filters

IP Chains

IP Chains is a packet filter for Linux systems. A packet filter compares the fields of each packet's header against a set of rules; these rules determine the fate of the packet, i.e., allowed, or dropped. Below is a brief overview of the IP Chains system. For a more in-depth explanation consult the Linux Documentation Project's IP Chains HOWTO [1].

Chains

The IP Chains system uses lists of rules called chains. There are three built-in chains; **input**, **forward** and **output**; which are used for incoming, forwarded and outgoing traffic, respectively. It is also possible to create custom chains that may be jumped to from any of the built-in chains.

Figure 2 shows how incoming packets are handled by a Linux system. After having their checksum verified and passing a sanity check, all packets are processed by the **input** chain.

If they are accepted; masqueraded packets are passed directly to the output chain; local packets are passed to the system; and forward packets are passed to the **forward** chain and then to the **output** chain.

When a packet is handed to a chain for processing, it is compared progressively against each rule in that chain. Each rule contains a *target* that specifies what is to be done to the packet if the rule is matched - common targets are **ACCEPT**, **REJECT** and **DENY**. The target may also be a custom chain, which causes the packet to be compared against each rule in that chain until a further match is made.

If a packet reaches the end of a custom chain processing returns to previous chain. If the packet reaches the end of a built-in chain, its fate is determined by that chain's *policy*, which will be to either: accept, reject or deny.

Rules

In IP Chains each rule specifies criteria a packet must match, and a target to jump to if that criteria is matched. The possible targets are: ACCEPT, the packet is either delivered to the system, or to either the **forward** or **output** chain, as appropriate; REJECT, the packet is dropped and an ICMP message is returned; DENY, the packet is dropped; MASQ, the packet is masqueraded as from this host; REDIRECT, the packet is redirected to a local port; RETURN, processing returns to the previous chain. The target may also be a custom chain.

The criteria to be matched are based upon the attributes of a network packets header. What can be matched depends upon the type of the packet: ICMP, UDP, or TCP. All packets can be matched on their source and destination addresses, as well as the protocol type of the packet. ICMP packets can be matched against the ICMP message type,

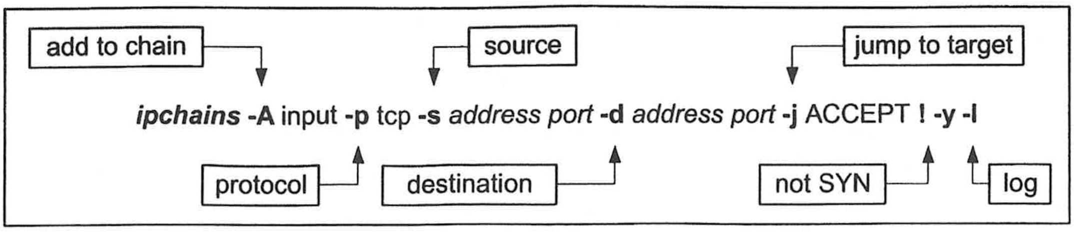


Figure 3: IPChains command

while UDP and TCP packets can be matched against their source and destination ports. TCP packets can also be matched against whether the SYN flag is set.

Commands

In IP Chains, rules are added to chains, and new custom chains are added, using the ***ipchains*** command. The specification of the command is as follows:

```
ipchains -[ADC] chain rule-specification [options]
```

Where "chain" is the chain the rule is to be added to, "rule-specification" specifies the criteria that must be matched, and "[options]" are various options that can be used, e.g., verbose mode ("-v").

Figure 3 shows an example of an IP Chains command. This command would add a rule to the **input** chain, which matches against a TCP packet that contains the source and destination address specified, and that is not a packet initiating a connection - does not have the SYN flag set. A matching packet would be logged, before continuing on.

Methodology

This section gives a step-by-step description of the methodology and produces a useable configuration script for an IP Chains firewall. This script should be run during system initialisation before any of the network interfaces are brought up.

Overview

When it is run, the script creates several custom chains, which along with the built-in chains, are then populated with rules. Network packets are mostly accepted or denied in the custom chains, with the rules in the built-in chains passing them to the custom chains for processing.

Figure 4 shows how incoming traffic flows through the chains. All incoming traffic is passed to the **input** chain. Each packet is then passed to each of the special chains, beginning with the **ingress** chain, until it is either denied or deemed to be valid. If the packet is destined locally, it is then passed to one of the local chains (one for each interface) until it is accepted; else it is logged and denied. Packets to be forwarded pass through these rules unmatched and are then accepted by forward rules, causing them to be passed to the **forward** chain.

When packets are received by the **forward** chain they are passed to either an incoming or outgoing chain as appropriate. These chains contain rules that match those packets that are to be forwarded by the packet filter; those that aren't are denied and logged.

In the **output** chain, packets that claim to be from the local host are processed first; sent to an appropriate chain for each interface, and if not accepted are denied and logged. Any remaining packets should be those that came from the **forward** chain. They are first sent to a chain that perform egress filtering, and then are double-checked by being resent to either the incoming or outgoing chain.

Example network and policy

The following example network is typical of those of organizational units or small businesses. An internal network is connected to an external public network; a properly configured firewall is required to control the traffic between the two networks. A good resource for determining appropriate rules for other protocols is the text "Building Internet Firewalls" [19].

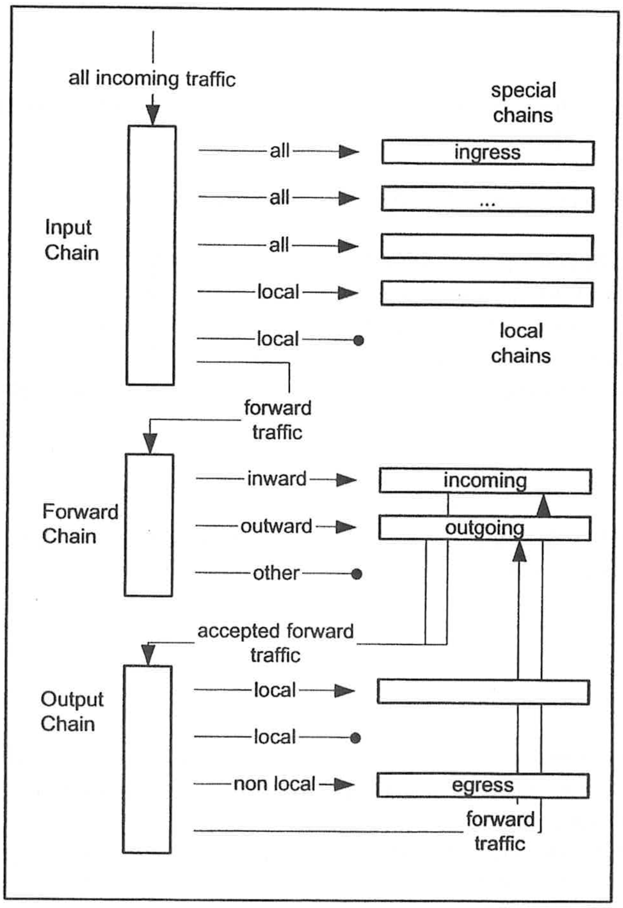


Figure 4: Graphical representation of final state

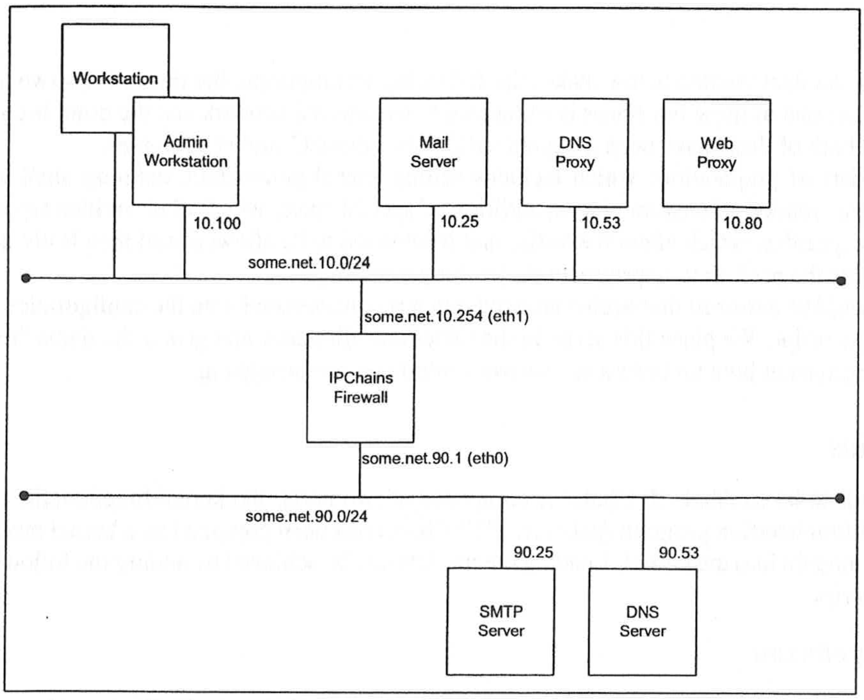


Figure 5: The example network

The network

The internal network contains a variety of machines including workstations, servers, and proxy servers. The only workstation that affects the firewall configuration is an administrative workstation used to perform remote administration of the packet filter. There are three relevant servers: a mail server, a DNS proxy, and a web proxy. We'll assume that these have been assigned identifiers corresponding the service they provide, e.g., the web proxy's address is some.net.10.80. The external network contains an SMTP server and a DNS server.

The network security policy

A network traffic policy should specify what traffic is allowed to flow within, into and out from, an organisational network. For brevity's sake the policy below only covers the traffic that contacts or crosses the firewall.

The network traffic policy for traffic from/to the firewall is as follows:

1. The firewall may send ICMP type 11 (time-exceeded) messages to the internal network.
2. The firewall may receive ICMP type 8 (echo) messages from the internal network.
3. The firewall may send ICMP type 0 (echo reply) messages to the internal network.
4. The firewall may send DNS packets to, and received DNS packets from, the internal DNS proxy.
5. The firewall may accept SSH connections from the Admin Workstation.

The network traffic policy for traffic over the firewall is as follows:

1. The firewall will forward incoming and outgoing ICMP type 3 (destination-unreachable) messages.
2. The firewall will forward incoming ICMP type 11 (time-exceeded) messages.
3. The firewall will forward incoming ICMP type 0 (echo reply) messages.
4. The firewall will forward outgoing ICMP type 8 (echo) messages.
5. The DNS proxy may send UDP datagrams to, and receive reply DNS datagrams from, the external DNS server.
6. The Web Proxy may make connections using http to any external web server.
7. The Mail Server may make SMTP connections to, and receive SMTP connections from, the external SMTP server.
8. All internal workstations may make SSH connections to any external machine.

Step-by-step

The methodology that we demonstrate below makes the following assumptions: the firewall has two network interfaces – no more, no less; one of these interfaces is connected to an external network and the other is connected to an internal network; and both of these have been assigned valid public class C subnet addresses.

The process consists of preparation; which includes setting kernel parameters, defining shell variables, and setting default policies; creation of custom chains; addition of special rules, which filter various types of improper traffic; addition of accept rules, which allow the traffic that is intended to be allowed; and then lastly the addition of "glue" rules, which pass the packets to appropriate chains for processing.

As you read, it should be assumed that each command given is concatenated onto the configuration script, giving at the end the complete script. We place this script in the "/etc/rc.d" directory and give it the name "rc.ipchains". It is set to execute during system boot up before any network interfaces are brought up.

Enabling IPChains

Firstly IP Chains needs to be enabled. IP Chains is composed of two parts: the kernel functionality that processes the packets, and the administration program *ipchains*. If IP Chains has been compiled as a kernel module it may be necessary to load the module into the kernel. Loading the module can be achieved by adding the following command to the configuration script:

```
/sbin/insmod ipchains
```

It is not the intention of this paper to describe how to do the installation of IP Chains; if you need further information, in this regard, you should consult the Linux Documentation Project's IP Chains HOWTO [1] and Firewalling HOWTO [20].

Setting Linux kernel parameters

There are some Linux kernel parameters that may be set to increase the effectiveness of the packet filter [21]. Most important, is enabling IP forwarding.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

The following will cause a packet to be dropped if a reply would leave through a different interface to the one it was received on.

```
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do

echo 1 > \$f

done
```

Source routing may be disabled.

```
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do

echo 0 > \$f

done
```

Also, directed broadcasts can be disabled.

```
echo 1 > icmp_echo_ignore_broadcasts
```

Interface information

To make later commands easier to read and maintain, information about each interface is stored in shell variables.

```
#  Public interface information
ETH0=eth0
ETH0_MASK=255.255.255.0
ETH0_NETID=<external net id>                ## eg. 192.168.33
ETH0_IP=$ETH0_NETID.1
```

```
ETH0_NET=$ETH0_NETID.0/$ETH0_MASK
ETH0_BCAST=$ETH0_NETID.255

#  Private interface information
ETH1=eth1
ETH1_MASK=255.255.255.0
ETH1_NETID=<internal net id>                  ## eg. 192.168.33
ETH1_IP=$ETH1_NETID.254
ETH1_NET=$ETH1_NETID.0/$ETH0_MASK
ETH1_BCAST=$ETH1_NETID.255
```

We will also define the following variables.

```
PUBLIC="! $ETH1_NET"
IPCHNS=/sbin/ipchains
```

In particular, note that **ETHx_NETID** only represents the first 3 octets of an network address. Hence **$ETH0_NETID**.255 would be the broadcast address of the first interface.

Setting default chain policies

The policies of the built-in chains should be set to a safe default. This ensures that if there is a problem with the rest of the configuration that packets will be dropped by default.

```
$IPCHNS --P input   DENY

$IPCHNS -P forward DENY

$IPCHNS -P output  DENY
```

Flush existing rules

We now flush existing rules.

```
$IPCHNS -flush
```

Create custom chains

Here we create the chains we presented in figure 4 that we will be populating. Firstly we create the chains for the special rules.

```
$IPCHNS -N ingress      2> /dev/null

$IPCHNS -N bcast        2> /dev/null

$IPCHNS -N undef        2> /dev/null

$IPCHNS -N reserved     2> /dev/null

$IPCHNS -N mcast        2> /dev/null

$IPCHNS -N private      2> /dev/null

$IPCHNS -N egress       2> /dev/null
```

A chain is created for incoming and outgoing local traffic on each interface.

```
$IPCHNS -N inLO         2> /dev/null

$IPCHNS -N in$ETH0      2> /dev/null

$IPCHNS -N in$ETH1      2> /dev/null
```

```
$IPCHNS -N outLO        2> /dev/null

$IPCHNS -N out$ETH0     2> /dev/null

$IPCHNS -N out$ETH1     2> /dev/null
```

Also a chain is created for each direction of forwarded traffic.

```
$IPCHNS -N $ETH0$ETH1   2> /dev/null

$IPCHNS -N $ETH1$ETH0   2> /dev/null
```

As flushing IP Chains using **–flush** doesn't delete custom created chains, possible error output, from creating an already existing chains is send to **/dev/null**. Note that chain names have a maximum length of 8 characters, thus if you give a chain a name longer than this, attempting to add a rule will cause errors.

Special chains rules

Firstly, the rules that protected against malicious packets are added to the special chains.

Ingress

The following rules perform ingress filtering. This rule protects against traffic that pretends to be from the loopback address, but isn't.

```
$IPCHNS -A ingress -i ! lo -s 127.0.0.0/24 -j DENY -l
```

Next a rule is added for each network interface. The first rule denies and logs packets that were received on the external interface but claim to be from the internal network; the second denies and logs packets that were received on the internal interface but claim to be from a public address.

```
$IPCHNS -A ingress -i $ETH0 -s $ETH1_NET -j DENY -l

$IPCHNS -A ingress -i $ETH1 -s $PUBLIC -j DENY -l
```

Broadcast

The following rules prevent broadcast packets from being forwarded or accepted by the firewall. The first two rules deny and log packets that are using the broadcast addresses improperly, i.e., using the destination address as the source address and vice versa.

```
$IPCHNS -A bcast -s 255.255.255.255 -j DENY -l

$IPCHNS -A bcast -d 0.0.0.0 -j DENY -l
```

The following rules simply deny all broadcast packets. Due to the difficulty of differentiating between valid and invalid broadcast traffic, these packets are not logged.

```
$IPCHNS -A bcast -s 0.0.0.0 -j DENY

$IPCHNS -A bcast -d 255.255.255.255 -j DENY

$IPCHNS -A bcast -d $ETH0_NETID.0 -j DENY

$IPCHNS -A bcast -d $ETH0_NETID.255 -j DENY

$IPCHNS -A bcast -d $ETH1_NETID.0 -j DENY

$IPCHNS -A bcast -d $ETH1_NETID.255 -j DENY
```

We also want to disable loop-back broadcast traffic.

```
$IPCHNS -A bcast -d 127.0.0.0 -j DENY

$IPCHNS -A bcast -d 127.255.255.255 -j DENY
```

Undefined, Reserved and Multicast addresses

The following commands protect against routing packets that contain non-routable addresses. These rules are added to the **undef**, **reserved**, and **mcast** chains.

Later, when the packets are processed by these chains in the **input** chain, order is important, as the mask used for multicast traffic also matches addresses matched by the masks for reserved and undefined addresses. It would be possible to combine all these rules into one rule, however leaving them as separate rules in separate chains makes their log entries easier to identify. Receiving packets containing undefined addresses is more likely to be of concern than receiving multicast packets.

First we add the rules to detect undefined addresses.

```
$IPCHNS -A undef -s 248.0.0.0/5 -j DENY -l

$IPCHNS -A undef -d 248.0.0.0/5 -j DENY -l
```

Next we add the reserved rules.

```
$IPCHNS -A reserved -s 240.0.0.0/5 -j DENY -l

$IPCHNS -A reserved -d 240.0.0.0/5 -j DENY -l
```

Lastly we add the multicast rules.

```
$IPCHNS -A mcast -s 224.0.0.0/4 -j DENY

$IPCHNS -A mcast -d 224.0.0.0/4 -j DENY
```

Private Address Ranges

These rules block traffic to and from private subnets.

```
$IPCHNS -A private -s 10.0.0.0/8 -j DENY -l

$IPCHNS -A private -d 10.0.0.0/8 -j DENY -l

$IPCHNS -A private -s 169.254.0.0/16 -j DENY -l

$IPCHNS -A private -d 169.254.0.0/16 -j DENY -l

$IPCHNS -A private -s 172.16.0.0/12 -j DENY -l

$IPCHNS -A private -d 172.16.0.0/12 -j DENY -l

$IPCHNS -A private -s 192.0.2.0/24 -j DENY -l

$IPCHNS -A private -d 192.0.2.0/24 -j DENY -l

$IPCHNS -A private -s 192.168.0.0/16 -j DENY -l

$IPCHNS -A private -d 192.168.0.0/16 -j DENY -l
```

Allowing Services

We now add the rules that implement the policy set forth in the network security policy.

Local Services

Rules for local services are added to the chains with the ***in*** and ***out*** prefix. For each incoming connection a connection rule is added to the appropriate *in* chain, and a reply rule is added to the appropriate *out* chain. Similarly the reverse happens for each outgoing connection.

To implement policy number (1), which allows ICMP time-exceeded messages to the internal network.

```
$IPCHNS -A out$ETH1 -p icmp --icmp-type 11 -d $ETH1_NET -j ACCEPT
```

To implement policy number (2), which allows ICMP echo messages from the internal network.

```
$IPCHNS -A in$ETH1 -p icmp --icmp-type 8 -s $ETH1_NET -j ACCEPT
```

To implement policy number (3), which allows ICMP echo reply messages from the firewall to the internal network.

```
$IPCHNS -A out$ETH1 -p icmp --icmp-type 0 -d $ETH1_NET -j ACCEPT
```

To implement policy number (4), which allows the firewall to make DNS requests, the following rules are added to the **out$ETH1** and **in$ETH1** chains.

```
$IPCHNS -A out$ETH1 \
  -p udp -sport domain -d $ETH1_NETID.53 domain -j ACCEPT

$IPCHNS -A in$ETH1 \
  -p udp -s $ETH1_NETID.53 domain -dport domain -j ACCEPT
```

To implement policy number (5), which allows an administrative workstation to connect to the *ssh* service, the following rules are added to the **in$ETH1** and **out$ETH1** chains respectively.

```
$IPCHNS -A in$ETH1 \
  -p tcp -s $ETH1_NETID.100 -dport ssh -j ACCEPT

$IPCHNS -A out$ETH1 \
  -p tcp -sport ssh -d $ETH1_NETID.100 -j ACCEPT ! -y
```

Forwarded Services

For packets that are to be forwarded from one interface to the other, rules are added to either the **$ETH1$ETH0** or **$ETH0$ETH1** chain, as is appropriate.

To implement policy number (6), which allows incoming and outgoing ICMP destination-unreachable messages.

```
$IPCHNS -A $ETH0$ETH1 -p icmp --icmp-type 3 -j ACCEPT

$IPCHNS -A $ETH1$ETH0 -p icmp --icmp-type 3 -j ACCEPT
```

To implement policy number (7), which allows incoming ICMP time-exceeded messages.

```
$IPCHNS -A $ETH0$ETH1 -p icmp --icmp-type 11 -j ACCEPT
```

To implement policy number (8), which allows incoming ICMP echo reply messages.

```
$IPCHNS -A $ETH0$ETH1 -p icmp --icmp-type 0 -j ACCEPT
```

To implement policy number (9), which allows outgoing ICMP echo messages.

```
$IPCHNS -A $ETH1$ETH0 -p icmp --icmp-type 8 -j ACCEPT
```

To implement policy number (10) allow DNS we add.

```
$IPCHNS -A $ETH1$ETH0 -p udp \
  -s $ETH1_NETID.53 domain -d $ETH0_NETID.53 domain -j ACCEPT

$IPCHNS -A $ETH0$ETH1 -p udp \
  -s $ETH0_NETID.53 domain -d $ETH1_NETID.53 domain -j ACCEPT
```

To implement policy number (11) allowing HTTP traffic we add.

```
$IPCHNS -A $ETH1$ETH0 \
  -p tcp -s $ETH1_NETID.80 3128 -d $PUBLIC http -j ACCEPT

$IPCHNS -A $ETH0$ETH1 \
  -p tcp -s $PUBLIC http -d $ETH1_NETID.80 3128 -j ACCEPT ! -y
```

To implement policy number (12) allowing SMTP connections we add the following rules.
First we add the rules to allow outgoing connections to the external SMTP server.

```
$IPCHNS -A $ETH1$ETH0 -p tcp -s $ETH1_NETID.25 \
  -d $ETH0_NETID.25 smtp -j ACCEPT

$IPCHNS -A $ETH0$ETH1 -p tcp -s $ETH0_NETID.25 smtp \
  -d $ETH1_NETID.25--j ACCEPT ! -y
```

Then we add rules to allow the external SMTP server to connect to our internal one.

```
$IPCHNS -A $ETH0$ETH1 -p tcp -s $ETH0_NETID.25 \
  -d $ETH1_NETID.25 smtp -j ACCEPT

$IPCHNS -A $ETH1$ETH0 -p tcp -s $ETH1_NETID.25 smtp \
  -d $ETH0_NETID.25 -j ACCEPT ! -y
```

Lastly to implement policy number (13), allowing ssh connections to outside hosts we add.

```
$IPCHNS -A $ETH1$ETH0 -p tcp -s $ETH1_NET -d $PUBLIC ssh -j ACCEPT

$IPCHNS -A $ETH0$ETH1 -p tcp -s $PUBLIC ssh -d $ETH1_NET -j ACCEPT ! -y
```

Egress Filtering

The reason for egress filtering is to ensure that outgoing packets actually came from where they claim to have come from. This seeks to protect organizations from being used as staging areas for launching attacks against other systems.

Egress filtering is a variation of ingress filtering, which has already been performed in the **ingress** chain. The only case that has not been allowed for is when the firewall itself sends spoofed packets. On first thought one might think that if something is causing the firewall to send spoofed packets then chances are that it would also have sufficient privileges to remove any firewall rules intended to stop this. This is probably true, but assuming the possibility that it could be a worm sending the packets the following rules will attempt to stop them.

Note that packets originating from the firewall that have a valid source address and will leave out an appropriate interface have already been accepted at this stage.

The first rule stops the firewall from sending a packet to a external host with the spoofed source address of an external host. The second rule stops the firewall from sending a packet to an internal host with the spoofed source address of an external host.

```
$IPCHNS -A egress -i $ETH1 -s $ETH1_NET -j DENY -l

$IPCHNS -A egress -i $ETH0 -s $PUBLIC   -j DENY -l
```

Unfortunately, because it is not possible in IP Chains to differentiate between a packet that has been passed from the forward chain, and a packet that has originated from the local host, it is not possible to protect against the firewall sending a packet to an external address with the spoofed source address of an internal address.

One reason an attacker may do this is to hide the fact that the firewall has been compromised.

Linking the chains

So we currently have the following chains; the special chains, **ingress**, **bcast**, **mcast**, **private**, and **egress**; the local chains, **in$ETH0**, **out$ETH0**, **in$ETH1**, and **out$ETH1**; and the forward chains, **$ETH0$ETH1**, and **$ETH1$ETH0**. But at this stage no packets actually reach these chains, as the default built-in chains **input**, **forward**, and **output**, are currently empty.

It is necessary to add rules to these chains that pass the packets to the other chains for filtering. Also the ordering is important as well as packets should be processed by the special chains, then the local chains, and only lastly the forward chains.

The input chain

Jumping to the special chains

It is first necessary to jump to the special chains. These perform the task of specifically matching illegal, invalid, or unwanted broadcast and multicast packets, then dropping them. All packets must be processed by these six chains before being processed by the local chains.

```
$IPCHNS -A input -j ingress

$IPCHNS -A input -j bcast

$IPCHNS -A input -j undef

$IPCHNS -A input -j reserved

$IPCHNS -A input -j mcast

$IPCHNS -A input -j private
```

Handling Local Traffic

Next it is necessary to handle local traffic. This must be processed before traffic to be forwarded, as forwarded traffic may accept services not appropriate for the firewall. And as the firewall itself also has IP addresses that belong to the networks either side of it, it may mistakenly be included in a rule that accepts a connection.

Also if forwarded traffic is processed in the forward chain, then it needs to be accepted in the input chain first. This means that blanket acceptance of packets to cause them to move to the forward chain, will also cause those packets addressed to a local interface to be accepted as well.

Therefore for each local interface, if packets are destined for that IP address they are passed to the relevant local chain, where if they are to be accepted, the will be accepted. Any that aren't are dropped by the next rule.

Rules for the loop-back interface:

```
$IPCHNS -A input -i lo -d $ETH0_IP -j inLO

$IPCHNS -A input -i lo -d $ETH1_IP -j inLO

$IPCHNS -A input -i lo -d 127.0.0.0/8 -j inLO

$IPCHNS -A input -i -d 127.0.0.0/8 -j DENY -l
```

For the external interface:

```
$IPCHNS -A input -i $ETH0 -d $ETH0_IP -j in$ETH0

$IPCHNS -A input -d $ETH0_IP -j DENY -l
```

And for the internal inteface:

```
$IPCHNS -A input -i $ETH1 --d $ETH1_IP -j in$ETH1

$IPCHNS -A input -d $ETH1_IP -j DENY -l
```

For the rule that passes the packet to the custom chain we check the interface to make sure that the packet came in an appropriate interface for that chain.

Handling Forwarding Traffic

Next it is necessary to process packets to be forwarded. Forwarded packets need to be accepted by the input chain, which passes them to the forward chain, then accepted by the forward chain, which passes them to the output chain, then accepted by the output chain.

So you need to add a rule to the input chain to match packets to be forwarded. These packets can either be accepted, causing them to be passed to the forward chain, or passed to the appropriate $ETHx$ETHx chain, in which case if they are allowed they will be passed to the forward chain.

```
$IPCHNS -A input -i $ETH0 -d $ETH1_NET -j ACCEPT

$IPCHNS -A input -i $ETH1 -d ! $ETH1_NET -j ACCEPT
```

Deny

At the end of the input chain we add a rule that denies and logs any remaining packets.

```
$IPCHNS -A input -j DENY -l
```

The forward chain

Next we add rules to the forward chain that are similar to those of the **input** chain except that they now pass the packets to a chain appropriate for the direction of traffic. Note that in the **forward** and **output** chains, the '-i' argument means the output interface.

```
$IPCHNS -A forward -s $PUBLIC -i $ETH1 -j $ETH0$ETH1

$IPCHNS -A forward -s $ETH1_NET -i $ETH0 -j $ETH1$ETH0
```

The ingress filtering means that we can trust the source and destination addresses so we could also use these rules.

```
$IPCHNS -A forward -s $PUBLIC -d $ETH1_NET -j $ETH0$ETH1

$IPCHNS -A forward -s $ETH1_NET -d $ETH0_NET -j $ETH1$ETH0
```

We also add a rule that denies and logs any remaining traffic.

```
$IPCHNS -A forward -j DENY -l
```

The output chain

With the input chain, all packets can be processed as though they have just been received by the box. However, with the output chain, packets may be coming from either the forward chain, or may have originated from the firewall. For this reason local traffic is processed first. Then packets are sent to the egress chain. Then any forward traffic is accepted.

Local traffic

Local traffic is processed first by filtering out traffic sourced from a local address - loopback or local interface. The outgoing interface is checked to only match packets that will be going out an appropriate interface; effectively performing egress filtering on those packets. Inappropriate packets are denied and logged.

```
$IPCHNS -A output -i lo -s $ETH0_IP -j outLO

$IPCHNS -A output -i lo -s $ETH1_IP -j outLO

$IPCHNS -A output -i lo -s 127.0.0.0/8 -j outLO

$IPCHNS -A output -s 127.0.0.0/8 -j DENY -l

$IPCHNS -A output -i $ETH0 -s $ETH0_IP -j out$ETH0

$IPCHNS -A output -s $ETH0_IP -j DENY -l

$IPCHNS -A output -i $ETH1 -s $ETH1_IP -j out$ETH1

$IPCHNS -A output -s $ETH1_IP -j DENY -l
```

Egress filtering

Next we perform egress filtering on the remaining outgoing packets - those that have come from the forward chain - by jumping to the **egress** chain.

```
$IPCHNS -A output -j egress
```

Forward traffic

At this point any packets should be forward packets; we double check by jumping to the appropriate forward chain.

```
$IPCHNS -A output -s $PUBLIC -i $ETH1 -j $ETH0$ETH1

$IPCHNS -A output -s $ETH1_NET -i $ETH0 -j $ETH1$ETH0
```

Deny

Finally we add a rule that denies and logs any packets that have survived thus far.

```
$IPCHNS -A output -j DENY -l
```

Future Directions

The methodology presented does not consider the use of private networks, masquerading, or the possibility of connecting more than two networks. These areas will be addressed; and a similar methodology will be produced for the IP Chains successor IP Tables [22].

Conclusion

While firewall rule sets are incredibly complex, a thorough methodology for developing the rule sets can increase their quality tremendously. We have discussed problem traffic types that need to be considered when developing a firewall configuration and have presented a methodology for developing rules sets that take these into account.

Acknowledgements

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

References

[1] P. Russell, "Linux IPCHAINS HOWTO". Version 1.0.8, July 2000.
`http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html`

[2] Standard ISO 7498 "Information Processing Systems - Open Systems Interconnection - Basic Reference Model". Reference Number: ISO 7498:1984(E), ISO TC97/SC21 Secretariat, ANSI, New York, New York, 1984.

[3] M.J. Ranum, F. M. Avolio, "*A Toolkit and Methods for Internet Firewalls*".
In Technical Summer Conference, pages 37–44, Boston, Massachusetts, June 1994. USENIX.

[4] M. Bykova, "Statistical analysis of malformed packets and their origins in the modern Internet". March 2002.
`http://irg.cs.ohiou.edu/papers/`

[5] CERT Advisory CA-1996-21, "TCP SYN flooding and IP spoofing attacks".
`http://www.cert.org/advisories/CA-1996-21.html`

[6] Ping of death. `http://www.insecure.org/sploits/ping-o-death.html`

[7] CERT Advisory CA-1996-01, "UDP port denial-of-service attack".
`http://www.cert.org/advisories/CA-1996-01.html`

[8] S. Whalen 2001, "An introduction to ARP spoofing". Revision 1, April, 2001
`http://chocobospore.org/projects/arpspoof/arpspoof.pdf`

[9] CERT Advisory CA-1995-01, "IP spoofing attacks and hijacked terminal connections".
`http://www.cert.org/advisories/CA-1995-01.html`

[10] P. Ferguson, D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing". RFC 2827, May 2000.

[11] Nmap. `http://www.insecure.org/nmap/`

[12] R. Droms, "Dynamic host configuration protocol" RFC 2131, March 1997.

[13] CERT Advisory CA-1998-01, "Smurf IP denial-of-service attacks".
`http://www.cert.org/advisories/CA-1998-01.html`

[14] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, "Address allocation for private internets". RFC 1918, February 1996.

[15] CERT Advisory CA-1996-26, "Denial-of-service attack via ping".
`http://www.cert.org/advisories/CA-1996-26.html`

[16] O. Arkin, "ICMP Usage in Scanning: The Complete Know How".
`http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf`

[17] N. Freed, "Behaviour of and requirements for Internet firewalls". RFC 2979, October 2000.

[18] R. Braden, "Requirements for Internet hosts: communication layers".
RFC 1122, October 1989.

[19] D. B. Chapman, E. D. Zwicky, "Building Internet Firewalls". O'Reilly & Associates, Inc., 1995.

[20] M. Grennan, "Firewall and proxy server HOWTO". Version 0.80, February 2000.
`http://www.tldp.org/HOWTO/Firewall-HOWTO.html`

[21] T. Bowden, B. Bauer, J. Nerin, "The /proc filesystem". November 2000.
`http://hr.uoregon.edu/davidrl/Documentation/filesystems/proc.txt`

[22] IP Tables. `http://www.iptables.org/`

Hello World! The Business of Doing Software

Mark White
Apviva Technology Partners
`mark@apviva.com`

Introduction

Anyone who has written a line of code will agree that software development is both a rewarding experience and an extremely demanding profession. To some, source code generation is a near-effortless process, while others find it a nightmare of blood, sweat and tears. However, the creative side of software development – try naming another profession so like alchemy, where the generation of something of substance from nothing more than your mind and fingers – has proven for many to be both attractive and rewarding. Especially when you can see the results of your good work – happy, productive users of technology, wholly enabled by your skills, talents, and occasional ability to touch-type.

Software can be a very efficient business proposition. We feasibly create "bits" out of practically nothing and replicate them for sale cheaply and often. A software development business *can* be operated with high-margins. (Note my careful use of the word "can" – indeed *any* business can also be operated unsuccessfully. Followers of the US and European capital markets lately will specifically get my meaning!) And it can also be *rewarding* (yes, more italics!) in many, many ways.

To be absolutely clear, our primary interest here is in ensuring the "rewarding" part of building a software business translates into *financially or commercially rewarding* for its shareholders. There are many other qualifiers we could apply to "rewarding", and I would hope that they are not all incompatible. In reading this, you are possibly already an experienced software developer. If you are, and have spent some time in the industry, you are probably acutely aware that successful software development does not necessarily equate to a successful software business. In this paper, we will try and condense the myriad of issues associated with starting and building a successful software business down to just seven basic principles.

To complete the introduction, an important disclaimer - *caveat emptor*. I'm neither a lawyer, accountant, physician nor magician. And I certainly don't claim to be in the pages that follow. To build a successful software business, you will need to draw on advice from all of the above, and then some. This paper, hopefully, will serve as a sort of starting point for your journey.

Principle #1: Know What to Build

At first glance, Principle #1 probably elicits as "D'uh!"-like response. That's the whole point of developing software – knowing what it is you need to create, what problem you need to solve, what colour to make your GUI buttons etc. But in reality, software developers are unfortunately mostly told what to build by others – system analysts/architects/idiot users etc. – and don't always get given the proverbial clean slate on which to work.

The foundation of this principle is that you should have some idea of a problem domain, a neat piece of analytical alchemy or other "thing" that you need to solve. Some ideas result from working on other problems, or maybe the late night creativity bug strikes and you run to your PC and produce C code like fireworks. Perhaps you notice a particularly inefficient business process at work and think "Hey, I wonder if we could fix that..."

Remember, someone once decided to create a visual numerical calculation tool. To be able to perform these tasks interactively and iteratively without requiring programmers and batch runs was a novelty. They didn't know at the time that it would eventually be called a "spreadsheet"!

At this point, you should be able to write down three or four ideas or concepts of ways in which new software could make the world a better place. If you can't, go read the introduction again, and/or work as a computer

programmer for a few more weeks. There are, and always will be, problems that can be solved with software. Your challenge is to decide on one that you can prove, demonstrate, and or complete before anyone else does it better!

Principle #2: Know How to Build It

To some, Principle #2 is a done deal. (If you are a C++ guru, you will probably have the base object classes coded on your notebook before my presentation is finished!) However it's often helpful to take a step outside of your own technology comfort zone before you make a final decision.

Try asking yourself a couple of questions:

- Is this going to be valuable as a software product or a provided service?
- Is this really the best/most cost effective platform to deliver this software?
- If it can't be done on this platform/language, what would an alternative be?

The list above is not exhaustive, but should give you an idea of what's essentially devils advocacy. Try and ensure that you are seeing this from outside your own comfort zone. Ask advice: people you know, prospective users, and other folks with opinions.

On to one of my favourite topics: licensing. You really should develop a strategy for this before you cut a single line of code. Together with a later principle (#4), any potential for creating a viable software business or product really mandates that you have licensing under control.

I will in no way declare any bias or tendencies toward or against various open source licenses. You should, however, seek to be or *become extremely aware of the rights and/or limitations various licensing regimes impose on both developers and users/customers* of your proposed software product(s).

Commercial licenses are quite common in the business software world. They typically impose no-copy, no-redistribution, no-modification and other restrictions on the user, and indemnify the developer/licensor against any bad things that may happen. Source code access is typically not provided under commercial licenses.

In many cases (e.g. Microsoft, Oracle), commercial licensing will be the most obvious strategy. But if considering deployment on or use of open source platforms and libraries, it is imperative that you can distinguish between, say, the GPL and Library (or Lesser) GPL with respect to restrictions placed on software linked/deployed on such libraries.

As it's 2002 and by attendance at an AUUG conference you are possibly displaying at least a peripheral interest in open source software, it would be worthwhile to highlight that OSS licenses do not necessarily prohibit commercial success. It just promotes a favourite concept of mine:

"Innovation in software doesn't end at development, but instead at deployment and use."

If your value proposition to users is simply the redistribution of free code, be prepared for a similar fate to [insert names of nearly every Linux company here]. Instead, consider:

- What really is your product?
- How is value delivered from the customer's perspective?
- Does your use of OSS give you a competitive advantage in delivering that value?
- Can you sustain that competitive edge, even though anyone else can (theoretically) download the same code?
- Can you *really* make money from that?

There is no magic to be discovered in creating a commercial business out of open source software. It's just another business; it's managed as a business and ultimately will be evaluated as a business by stakeholders. There is of course a mystery surrounding the *sustainability* of such businesses – any takers?

By this stage, you should not only have an idea of what your software product should do, you should also have objectively decided on the language, platform(s) and licenses under which you can build it. (And yes, you will still get extra credit if you have completed any code by now!)

Principle #3: Stop Thinking Like a Software Developer

This may come across as blasphemous, but we really need to get over the fact that software developers indeed do not know everything there is to know. In fact, in many real-world situations we simply aren't the smartest people in the universe!

Principle #3 is really about ensuring you can develop and maintain a Zen-like focus on what you are trying to achieve with your software. If you want to build the world's most scalable, fastest batch-processing end-of-month system, that's great – but what if the problem really is Joe User having to wait 4 minutes to enter data into an illogically laid-out input screen? Are you simply building a better mousetrap, and, if you are, have you *noticed* any mice lately?

The job of a software developer is in many instances quite simple: here's a specification, build it, test it, and go home at 5pm. If it breaks, stay back late and fix it.

But true innovation in the software business comes from applying finely honed analytical skills to problems in the real world, and then implementing an efficient/better solution. If you start conceptually with the solution, you'll likely be chasing a problem that may not exist.

Some developers may find the application of Principle #3 is testing. Keep at it. Seek advice, query those around you, and find domain experts from whom you can learn. Clearly define the problem, devise a solution, then (and only then) start the iterative process of review.

Make sure you can quantify the problem – not in lines of code, but in cost or time (on the basis of time = money). How much does it cost Company X if service orders take 10 minutes to process? How much money can they save by outsourcing mail servers? What about the implications to a corporate marketing plan if you can devise an e-mail system that tracks responses even 10% more accurately? Can your product do the job of product A, B or C at 50% of the price?

Always return to your definition of the problem, as you will gain insight into the true nature and scope of the problem through revisiting it over and over again.

Lots of successful business people can't and will never become good programmers. The reverse simply isn't true, however you do have to know when to think in accordance with each discipline.

Principle #4: IP is not just a Network Protocol

Having a clear position on all Intellectual Property (IP) assets is a critical element in the business of software. You may *think* you own a piece of code, and as it's sole author, you may have a sensible reason to feel that way. But there will be numerous ways in which others may have a valid claim on your work. It is always better to identify possible claims before they are made, and if possible before any code is written or even ideas are documented.

To illustrate by way of example, consider a University student – say at honours or masters level. In most circumstances, she or he will be completely unencumbered by their institution with respect to ownership of any innovations or results they produce in the undertaking of either course or project work. This is a good thing! It means they can create freely, and own whatever they produce. This is a solid IP foundation.

But say the above student takes vacation employment with their University department, perhaps to work as an assistant to a full-time researcher. The work they undertake may well be completely divorced from their own innovations, but the University provides a desk, a workstation and some bandwidth. In their spare time, said student devours pizza at work and hacks a little on their own program. Is this still a "solid" IP foundation? Unfortunately, no!

This example should demonstrate how easily the waters become muddy in relation to IP. The student undertaking vacation work is indeed using University resources supplied for a specific purpose to undertake "other" work. While money has not been exchanged for the software, the provision and utilisation of resources potentially gives the University grounds for making a claim on the software.

Another, possibly simpler rule applies to nearly any software developer who has signed an employee contract. Read it again: you will probably find that you can't so much as have a creative dream at home without it becoming the property of your employer. People however still write their own software, so there are ways to address such a situation. But you must always be conscious of obtaining appropriate permissions, disclaimers and releases as soon as possible.

In all of these cases, a simple rule to determine your IP position is "follow the money." If someone else contributed financially in some way to the process of software development – either directly or in kind – there's scope for a claim. If that's the case, consider your position and consult with a legal professional. Often, the possible claimant may have no interest in your software, and may be perfectly happy to sign a document disclaiming any ownership or rights to it. Or maybe they won't – that's why we have lawyers.

Hopefully you will reach a point where you have ascertained either partial or full ownership over your code, or are working for/with someone who does. IP management is a continuous component of the software business. On-going IP management is an extremely complex area, but here are a few hints:

- CVS logs are your friends. Even as a lone developer. Use them well, back them up, archive them, and make printed copies regularly. There will come a day when they are needed to defend your IP position

- Pedantically check and verify the licensing of any and all components you may incorporate as libraries, code fragments, applets, icons etc into your software. Even donations from your friends ("here, try this locking code"). Wherever possible, get some sort of documentation from the provider to disclaim any subsequent ownership. Keep logs of where and when you were given or downloaded the code.

- Software patents have the potential to make this world a scary, scary place. Even if you just dream up a cool way of doing something conceptually, write it down, sign, date it, and add it to your IP register (the same folder you keep your printed CVS logs).

- Non-disclosure agreements (NDAs) are pretty worthless tools, but unfortunately that's what we in the software industry are stuck with. Use them. "IP register" them too.

- Find a good lawyer who understands technology IP, before you really need one. Because in any software related business, one day you will.

- IP management is important to the business of software, but it shouldn't inhibit the process of innovation. It's just a necessary evil that, on the other hand, should serve to protect us. Create and adopt a strategy that works for you, and ensure you seek professional advice.

Principle #5: Become Your Own Worst Enemy

From the very moment you produce even a proof-of-concept demonstration, you should be capable of actively taking a contrarian view to every single idea, concept or premise about your nascent software product. Does it <u>really</u> work? Will it <u>really</u> scale? And does it really, truly solve the problem you are trying to solve, or does it just create different problems?

Most importantly, will perfect strangers <u>really</u> give you money to use it? And how will you find them?

Principle #5 extends way beyond having someone else test your code. Be prepared to regularly defend nearly every single decision you've made about design, architecture, applicability, pricing, target audience, platform. Because you can bet that other parties – with considerable influence on your success or otherwise as a software business – will challenge your stances on all of the above.

The good news is that you can readily find people with expertise, skills, and or experience to bounce ideas off. Forums like AUUG are sensational for people networking, but you could also check out other groups like Software Engineering Australia, and various business incubators that have links with professional firms (accountants, lawyers, marketing types etc). The "suits" aren't necessarily all bad. Just watch out for the bad ones!

Like IP management, Principle #5 should really evolve into a core and iterative process of any software business. A management team that isn't regularly validating their space, position, pricing, performance etc. isn't worth the paper their payslips are printed on. Even if you are the sole member of the management team, challenge yourself and your thinking regularly. And be prepared to take corrective action.

Principle #6: Code Matters

In any software business, the rush of innovation or quick prototype development should very quickly give way to a structured, managed development process. This is of course because that the code, as the significant asset you are trying to leverage, matters.

The primary challenge here is in identifying a way to make that transition from idea to product to maintenance chore as efficiently as possible. It often takes different people, skills and attitudes – but the whole point of software is in making money from people using your code, over and over again. So maintenance of that asset through a managed engineering process is a critical factor in the success of software companies.

On a similar note, quality is king in the software game. As an industry, we get a deservedly lousy reputation on this issue, and as a software entrepreneur, this is your best (and probably only) opportunity to ensure you start on a solid footing. Document your code. Maybe even add some comments. Test it to within an inch of your life, and when you are done, test it some more. Be ready and able to prove functionality in adversity, not just in ideal situations. If you set/publish performance (or other) metrics, make sure you can easily exceed them.

Most importantly, develop a plan, a set of processes, and manage development to that plan. Hire or engage the most talented people, the most diligent and trustworthy folks you can imagine to work on or support your code. It really is the most critical investment you can make in the business of software.

Having said all that, it is time to introduce a corollary to Principle #6, namely:

Principle #6a: Code Doesn't Really Matter

Sorry, but for some of you this is the bad news principle. If you are a software business, the "business" word matters a whole lot more than the "software" word. This is an absolute truism, so deal with it. Be prepared to fully consider your business as effectively divorced from the products and profession you have laboured to perfect.

This principle can be brutal to those who love their code. It is something you have built, crafted, created and nurtured. But in reality, it's a business asset, much like the telephone or your bookcase. It's what you do with that asset to generate money that matters.

Investors – and if you are building a business around your software, this includes you – really don't care if they are in the shoe or petroleum business. What concerns them on a daily (if not minute by minute) basis is the effectiveness of the business in creating revenue faster than it burns their investment dollars. Similarly, you need to really, really adopt Principle #5 at all stages in the business. If you don't, you will quickly find that a multitude of naysayers will. When you need to evangelise your technology, do so with a marketing hat on – don't make it the sole basis for your decisions and actions.

In summary, be attached to your family, friends, and hobbies – in fact, anything other than the software you wrote. You will need to be absolutely prepared to make and indeed accept cold and clinical decisions independent of any emotional attachment to your software. Because in business, you can safely assume a zero care factor from anyone else – and you will need to act the same way.

Principle #7: Everything Else Does Matter

While this paper has been aimed squarely at software developers, I would hope that it seems apparent that the software is really only just the beginning, and indeed a small component of, a successful software business. This isn't entirely true – and I'm a former software developer myself – but it does reflect the reality that successfully building software does not guarantee building a successful business.

In trying to whittle down the sheer enormity of topics that could be covered in the eEverything else matters" principle, I want to imbue the reader with the concept that – while "suits" may consider you a "mere" software developer – awareness and participation in "everything else" will be at worst a terrific learning experience, and at best can literally save your success when things go sour. In my experience, software types often have analytic and information digestion skills that far outweigh their commercial counterparts. Really, really smart software types would benefit from, for example, learning more about the sales process and discipline; not because you may have career aspirations in that direction, but because a better understanding of other parts of the business really helps your ability to contribute to the business at large.

So to the "everything else" set of topics: *products*, *markets*, and how they come together to create *value*. While this clearly isn't everything else, it gives a sense of focus into how you might approach the whole process of building a business, which just happens to involve software.

Make a point of understanding what your *product* really is. If you think – in the context of a software business – that it's something you can distribute as a CD or tarball, you are very, very mistaken. A "product" really encapsulates everything to do with what you sell or provide to your customers, including how you make them aware of it (marketing), how you get it to them (distribution, supply chain), how you deliver and support it (engineering and support services) – in fact everything to do with the success of the software falls within the boundaries of product management.

Make no mistake – really, really good technology can and inevitably will fail commercially because the "product" piece gets messed up. Unfortunately, the reverse does apply – and we've all been lumbered with using bad software as a result. Choose now which side you want to be on, and stick with it!

The *market* isn't something solely the responsibility of good-looking types in suits. It is all about understanding who will want or need your tools and why. It incorporates how and how much they will pay for the privilege of using your software to improve their lives. As an aside, in terms of a target audience, don't be limited by thinking only locally. Australia should factor into your business planning as a useful validation market. Export, and plan to export from day one.

Your ability to identify a market and successfully deliver a product into it creates *value* (and indeed, a successful software business). Value can be referenced in many ways; in nearly all of them a multiplier of the revenue (i.e. money derived) from the sale of your products is involved. Significantly, this is the model that investors readily apply when trying to calculate how much they want to invest in your company – and this paper is a long way from a guide to venture capital!

Another important way to reference value in your business revolves around the really, really important folks: your customers. Can you define this market/product value in such a way to easily and defensibly show true benefits from using your products?

When looking at the strategic side of business planning, always keep this (second) concept of value in mind. Remember, your success is entirely about providing discernable value to your customers – and even another dot com

bubble won't change that!

Conclusions

We have covered a lot of ground, and it's worth highlighting that we haven't really even scratched the surface of "how" a successful software business is created. The devil is and will always be in the details. What I have tried to provide is a high-level framework through which you can navigate the steps from software development to creating a business out of software.

In a way, I quite hope that by reading to this point you are having thoughts like "what a stupid paper, this is all just common sense!" That's the whole point – in developing software, and indeed in the commercial world – common sense is usually your best weapon, and often it's your entire arsenal. In creating software we can so often get drawn into the process of our own ingenuity, which occasionally clouds common sense. When this happens, go back and read these seven principles again. By all means be creative – that's what fuels great software. But in building a successful software business, never, ever lose sight of the simple things.

Australia has a much-admired history of producing outstanding software engineering talent. What we desperately need to develop now is more great software companies. I wish you well – and please send me e-mail with your experiences (positive or negative)!

Mark White is the co-founder of Apviva Technology Partners, an international market development consultancy. He is a current member of the AUUG board of directors, and an advisor to several Australian venture funds. When not consulting or writing from his home base in Brisbane, he's usually racing or training with the UQ Cycling Club.

References and Other (Possibly) Useful Links

- Open Source License Information `http://www.opensource.org/`
- The Strange Case of the Disappearing Open Source Vendors, by Tim O'Reilly `http://www.oreillynet.com/pub/a/network/2002/06/28/vendor.html`
- Software Engineering Australia `http://www.sea.net.au/`
- BITS Incubators: inQbator `http://www.inqbator.com.au/`, ADI `http://www.adinc.com.au/`
- AusIndustry `http://www.ausindustry.gov.au/`
- IP Australia Home Page `http://www.ipaustralia.gov.au/`

An Online Share Trading Platform in Seven Weeks – Extreme Programming and Open Source meets Godzilla

Ray Loyzaga
CommSecure Limited
ray@commsecure.com.au

Abstract

A case study of the use of open source systems and tools to develop an online equities trading platform is presented. The particular attributes of the problem are analysed in terms of their effect on the solution design. The chronology of the project is presented to demonstrate how some of the implementation techniques, such as prototyping and simulation, affected the success of the project and helped drive the aggressive development timelines.

Background

HSBC Australia is a small, but growing subsidiary of the world's second largest banking group. In early 1999 HSBC launched a relatively successful phone based discount share trading system (PhoneTrader). At that time the only retail bank with an Internet based trading platform was the Commonwealth Bank of Australia, but its back-office was largely paper based.

The management of HSBC had made the decision to launch an Internet based equities trading platform that incorporated "straight-through processing" (STP) and interfaced with their back-office settlement systems. In early 1999, HSBC engaged one of Australia's leading web design companies (one of the one's that thought that dark blue on a black background made the site look "cool"). Apart from basic brochure-ware, the site displayed a page that listed the previous trading days Top 20 most traded stocks. For very little change on a six-figure sum, they received a site that rendered poorly across a range of browsers and which had a curious 'feature' of listing a random stock 20 times on the Top 20 page.

It became apparent that a change of approach was in order …

Getting the Job

The initial goal was the replacement of the HSBC brochure-ware site with a site that would adopt the new look and feel that had been developed by HSBC and repair the Top 20 page.

This was seen as an initial test of our capabilities. The site was transferred to an Apache HTTP server and the Top 20 page was re-implemented with a very straightforward CGI program written in Python. Within a week the new service was up and running on an Intel PC running Linux and co-located at an ISP.

The system performed flawlessly; though the application was trivial, it was unbelievable what an effect the reliability of the service had on the bank management. It was clear that they had been deeply embarrassed by the daily Top 20 page problems. We had made it to Stage 2, but there were still a few battles to be won.

Bank Standards!

The bank had "standards" and needless to say "Open Source", Linux, Apache and Python did not feature. The bank standards are policed from Hong Kong, and we needed to convince them that all the FUD surrounding "Open

Source" based systems was baseless. We produced a paper highlighting the strengths of Open Source, with particular emphasis on reliability, performance, scalability, cost of ownership and flexibility. We all know the arguments and we were able to answer all of IT's "concerns", however that does not always means that you get the deal. In this case, the support of management was the lynchpin, they had confidence that we would deliver, and that our recommendations were the best path for them – they cleared the way for us.

Stage 2 – An online trading system

Having cleared most of the hurdles related to bank standards, we quickly established the following base technologies:

- Linux on Intel servers
- Python
- MySQL for any database requirements
- Apache

Why Intel?

An online equities trading platform can be broken down into some basic components:

- Data feeds
- Archival data and charts
- A messaging subsystem
- State management
- Presentation

A breakdown of the tasks, and therefore the computational requirements of each of these components, was undertaken to allow us to select the most appropriate hardware platform for this project. The prevailing opinion within the bank, and many other banks we have engaged with, is that these trading platforms require "big iron". Most retail banks have implemented these systems using hardware platforms that have cost many millions of dollars. This was a pretty easy argument to win within bank management because they just did not have that sort of money.

Data Feeds

The main data feed was the ASX SEATS interface; it provided the bulk of the data available to the system. The SEATS interface required very fast response, as each message may affect the prices that should be displayed. The message arrival patterns displayed very large peaks governed by things such as the opening and closing of the markets, major announcements, floats etc. Each message would require some data to be entered/modified within the equities databases.

While this might sound like it would require a lot of processing power, the "problem" is really:

- Receive/parse/digest ASCII data arriving at a peak of 128kbit/s
- Digesting the data really means modify a database table row
- There are about 1400 stocks on the ASX, so 1400 rows
- The size of a "row" is in the order of a few hundred bytes

When looked in this way, the problem can be seen to easily fit in the physical memory of a very modest machine. The actual data volumes don't really impose great stress on desktop systems. The problem would seem to benefit from a system that provides fast integer performance and fast memory. I/O performance would appear to be almost irrelevant.

Archival Data and Charts

Each day a small amount of data associated with each stock is stored to provide the input for a number charts that are used to analyse the performance of stocks. Once again, the total amount of data is not large. About 20 attributes are retained for each stock on a daily basis including the daily open, close, high and low prices and the volume of stock traded. Charts are generated from this data, with the number of data points being proportional to the timeframe being plotted and the number of stocks being compared. The computational complexity can vary depending on the type of analytic function and the number of data points being plotted. The typical CPU time required to generate a graph is in the order of a tenth of a second on an Intel PIII running at 800MHz.

This might appear to impose a fairly strong constraint on the throughput of the system, as only 10 chart requests per second would consume an entire CPU. Several factors provide relief from this apparent constraint, the most important being that most charts are totally historical. Once a chart is generated, the result is correct for the entire day, caching generated charts can provide almost complete relief from the computational burden of charts on a trading system. Another important factor is that chart lookups are relatively rare, usually comprising just a few percent of requests within an online trading environment.

Once again, with careful attention to the typical usage it can be seen that charting does not impose a burden on the system that cannot be met from a system that is built upon commodity CPUs. While caching is an important component to the solution, it does not mean that disk I/O performance imposes a major bottleneck. Charts are generally small in size to facilitate transfer to user browsers. This means that the total data required to cache the entire set of possible charts across all stocks is still only in the order of a few hundred megabytes. Consequently, a system with a few hundred megabytes of disk cache will provide a charting system that appears to perform at memory bandwidth speeds.

A messaging subsystem

At the core of many web based systems that interface to a number of data and service providers is a service that provides for the reliable switching of messages between the webservers and the back-end systems. The messages are typically small requests for particular information or actions. This message switch may provide an abstraction layer over the various formats employed by the back end systems, shielding the web applications from the vagaries of the back-end systems. This layer can also provide a security wrapper over the back-end systems to limit the operations available to the webservers. The main computational requirements for this component are the ability to deal efficiently with small amounts of (usually) textual data and to reformat the data for presentation to recipient systems. This performance of this component is mostly governed by integer CPU speed and memory bandwidth. As the messages are small, it is important that the system supports very efficient context switching to provide a low latency, high throughput message switching service.

State management

One of the areas that can cause bottlenecks in web based systems is the management of per-client state. Client state provides the information required to identify the user and the context of any operations they request. The size and volatility of per-client state can limit the available options available to for the implementation of the state management functions.

The client state must be available to application servers to provide the context, identity and permissions that will be in place during each operation. The "normal" solution is to provide a per-client "session" identifier that is used in conjunction with a database as an index to the stored per-client information. One of the problems with this approach is that the speed of the state database becomes a bottleneck for operations, since they all need to look up the data associated with the session-id. It has also been our experience that this approach encourages very little control of what is included in the per-client state. The larger the state, the larger the database, and this usually leads to a reduced database throughput.

An approach that depends upon state management within a single database reduces opportunities for horizontal scaling. Once the database is no longer able to achieve the throughput required, costs may rise dramatically as the only options available are moving to a much faster (and therefore more expensive) system or a much more expensive database which provides higher throughput by employing a distributed model. To avoid this problem some implementations use an associative client state, which binds a particular client to a particular database server. This provides an opportunity for horizontal scaling, at the cost of not necessarily spreading the load evenly over the servers.

In this particular case, we were very careful about what was stored in the per-client state and limited it to a sufficiently small amount of data (stored as a serialised Python dictionary) that we were able to pass as a cryptographically secured, compressed version of the state to the client with each reply. This meant that our "session-id" actually contained the data we required. The advantage of this approach is that no state database was required and the request could be sent to any server, providing low cost horizontal scalability.

Without the need for a high performance database, the state management task has been reduced to operations that work very well on a system that provides fast integer/memory performance.

Presentation

The presentation of data on an equities trading platform usually requires dynamic content on all pages. Dynamic content is usually driven through templates that provide the "look and feel" of the site. The main computational load is the manipulation of strings, the templates are easily contained in the system I/O buffers and so no great I/O performance is required.

By careful analysis of the main components of the trading platform and some judicious design decisions we were able to ensure that the system will be able to achieve high performance on relatively low priced commodity desktop servers.

Python

The entire system was written in Python, a very powerful interpreted object oriented language. We have found that Python provides a significant productivity advantage over some more commercially popular programming languages. This advantage can be as high as factors of five or more over languages such as Java, C++ and C. Despite being an interpreted language, Python provides "wrappers" around most of the common C libraries, which means that most computationally expensive operations are actually being performed at the native compiled code speed of the machine. Python's inherent object-oriented programming support and its clean syntax provides for a scalable and highly maintainable code base.

The types of applications we expected to require in the project would require efficient string handling, cryptography and database interfacing, all of which are either inherent strengths of Python, or provided to Python through very tightly integrated libraries such as OpenSSL.

MySQL

The database requirements of the project were pretty modest. Although the database would form a central storage point for historical data and the current state of the markets, the data would fit in the memory of the database server and the operations did not require true transactional semantics. This meant that existing open source databases were more than adequate for the task. At that time we chose MySQL as we had quite a lot of experience with it and we expected that it would provide the performance and reliability required. As "reads" were far more prevalent than "writes" on these databases, the performance of the databases could be greatly enhanced by indexing.

We presented a case for the use of all these technologies, and the management was strongly onboard. They were clearly impressed by our confidence that a commitment to these technologies will reduce costs (we gave them a ball-park figure), provide greater probabilities of success and greatly reduce the development timeframe. (We should have guessed this would be a two-edged sword!)

The Project Begins!

It was early June 1999 and after a week of waiting for a go-ahead for the project, we finally got called in for a meeting to tell us the "good news". They had successfully sold the project concept to the bank board, and to the regional management based in Hong Kong. That was the good news. The bad news was that the Managing Director of HSBC InvestDirect had mortgaged his "dangly bits" to get this approval - so we were not allowed to fail. There was much excitement all around and we were busily being introduced to some of the other IT staff within the bank that would participate in the development from the bank side. It was decided that we should find them a junior programmer who was skilled in Linux/Python/networking (I was working at Sydney Uni at the time, so I "procured" one of their better honours graduates). Then came the "bombshell", the "go-live" date was to be the 12th of August – about nine weeks away!

Implementation Approach

It was clear from the aggressive timelines of this project that a normal development process was not possible (in reality we had about seven weeks, since some time was required for commissioning the production servers, UAT and a software audit). A rapid application development (RAD) approach was to be followed where we attempted to produce, as quickly as possible, a set of prototype subsystems so that all parties could be able to build their systems

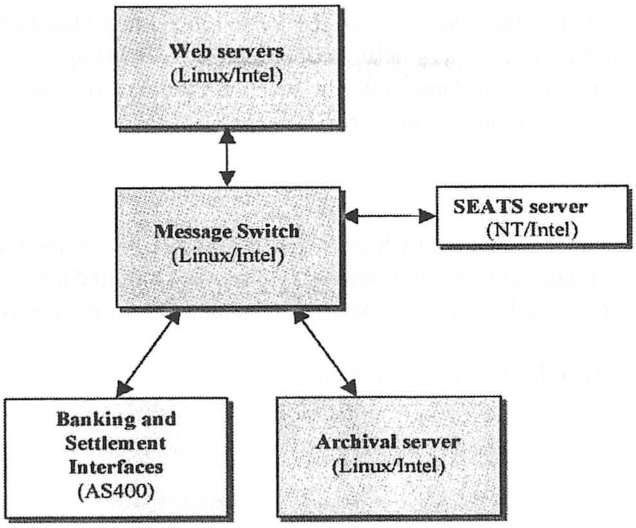


Figure 1: Design

and test against something that was simulating the behaviour of the server or client. It was important that this testing also provided the ability to characterise the peak throughput available from the subsystem.

HSBC was to supply some of their IT staff to the project, and they would mostly deal with the bank side servers that would need to be utilised for this service. The main system we needed to interface with was an AS/400 that ran the settlement system and interfaced with the banking system. Their best RPG programmer was available to code the interface that we developed between the trading platform and the banking systems. Communication to this system was to be provided via IBM MQseries controlled by a Java program written by HSBC. We were to talk to this Java server via a simple network socket based protocol.

HSBC wanted to handle the SEATS interface themselves, which they planned to host on an NT system, with the database being MS SQLserver. This system was to be written in Python by their newly acquired university graduate. HSBC also provided a project coordinator to keep the side of the project on track and their most senior IT manager was charged with clearing any obstacles that might be thrown in the project's way.

One of the project goals was that the system was to be maintainable by HSBC staff. By involving themselves in the project, they had first hand knowledge of all the design decisions and the major interfaces of the system.

We had a development team of three, with one person allocated to all of the web design and the HTML side of the system and the other two responsible for the underlying design, system interfaces and coding the various servers required to glue the system together.

Design

A rough outline of the important pieces was created, we were pretty lucky that part of the "selling" strategy was to create all of the web pages for all of the functionality. We had extended this a little further by providing a simple simulator for the stock exchange that allowed these pages to operate as they would when the system was live. This meant that all of the required functionality was understood and most of the messages that would support this functionality were very well defined.

The simulator gave us the confidence to believe that we had the important bases covered and the initial performance figures were very promising, despite all services running on the one desktop server. In the first few days of the project we were in a position to order the required equipment, document the interfaces and get all of the teams active on their part of the project.

Implementation

The initial phase of the implementation was centred around creating a number of test harnesses for each of the interfaces, so that each team had a simulated client or server for their part of the system. The message parsing code and handling of these test harnesses would be common to all of the core servers, if the simulators were presenting data wrongly to the developers, their repair would immediately improve the quality of the core message servers.

By the end of the second week, the message switch was in place. Its main job was to provide an abstraction layer over the various data servers. Where these services were provided by Linux/Python servers, the data was exchanged

using serialised Python dictionaries. For the other servers, the passed in Python objects were used to provide the data in the appropriate format, usually some form of delimited, ordered ASCII string.

The message server was driven from a configuration file, which would describe the all of the commands that were implemented. This file would contain lines of the form:

```
LOGIN MQserver 8189 JAVA
```

This meant that "LOGIN" messages are served by host MQserver port 8189, and the formatting required for the message was the "JAVA" style. For commands being served by systems that required a particular parameter ordering (such as the AS/400 which was connected via a JAVA MQseries handler) two files specifying the parameters and their order were required.

For the LOGIN command, the LOGIN.SEND file contained:

```
UserName
Password
DateTime
WebServerIP
```

This would provide enough context for the AS/400 to verify a login and to write an audit log. The LOGIN.RECV file would describe the expected response, which would include the status of the command, and if successful, the associated accounts that the user is able to operate upon.

The message server would check to see if the configuration file had changed and re-read it. This provided a very flexible way of extending the commands being supported as the project went on. As this message service was the central core of all operations, a lot of time was spent optimising its performance and ensuring that it was not exhibiting any memory leaks. Simulated loads of thousands of messages per second were run for extensive periods.

The SEATS connection was moving steadily, after initially setting up Python wrappers for the ASX supplied C libraries we tested out that we were able to perform some of the simple commands and receive data. From there the HSBC team set about creating the SQLserver tables that would store the data and writing the server that would have the task of receiving the ASX data and storing the appropriate fields into the database.

By the end of the third week of the implementation we were all pretty confident that we would be able to meet the schedule. The message switch had matured greatly and was being used to communicate to all servers. The simulators were slowly being converted to use whatever facilities were already completed, so more of the end-to-end system capabilities were becoming available.

Do we ever Learn?

Perhaps the feedback on the progress was just a little over enthusiastic. It served to encourage two major events, both driven by the Bank's business group. The first was the desire to extend the architecture to provide a minimum of two systems at each layer to provide increased reliability through redundancy. The second had the potential of providing a major coup for HSBC, the major retail banks were waking up to online broking and St George Bank was interested in rebadging the HSBC system as a fast way into the market. We needed to provide a quick response on the technical feasibility of these extensions and if possible include them into the delivery schedule.

Redundancy

Redundancy was addressed in a two-hour design meeting. Since we did not have much time to test any theories we decided that the best way forward was to map out all of the known messages and to see how they might be affected by the inclusion of multiple servers. We also needed to understand the effects on the servers that were already at a pretty advanced stage of development.

Since the per-client state was being provided with the request from the browser, it could be directed to any webserver so there was no problem with running a webserver farm, nor did we need to consider any client/server affinity. The webservers would make requests to the message switch, which would perform some request specific formatting and send the message to the appropriate server. Adding additional message switches would not require any major rewrites, the webservers just make a random selection from a list of available servers prior to sending the message. Fairly short timeouts are used to redirect to another server. If a server fails or times out, it is tried periodically until it is able to process transactions again, at which time it returns to the available servers list.

These changes only had minor effects on the server connection code, and the error handling/timeout code. We were careful to make the commands idempotent, which meant that we really didn't have to worry about any remnant state brought about by partially completed commands.

Allowing redundancy for some of the data sources meant changing the message switch to allow it to manage multiple servers for requests. Once again the same approach was used, with the only code effects occurring in the connection and error handling/timeout code. The configuration files were extended to allow multiple servers by allowing the specification of multiple lines for each command, which can be brought in and out of service by adding or removing lines from the file. Since the server automatically checks for changes, these additional servers can be brought into (or out of) service without restarting any components.

These changes provided a very simple way of allowing multiple servers, this not only shields the system from single point failures but provides a very simple performance upgrade path to ensure that the trading platform will be able to cope with the very sharp peaks in demand that are inherent in online share trading system. It also means that any successful "whitelables" like the St George opportunity will not require a radical redesign or totally different hardware systems to support a much larger user population.

The Home Straight!

The final weeks of the project got pretty hectic with most workdays ending around 2am. The commitment from the HSBC IT staff was exceptional and we were all working hard to make the deadline. The main area that was bringing problems was the performance of the SEATS interface system. The speed of that system would be critical to the overall performance of the whole platform. In addition, the system was exhibiting very poor performance when we simulated a "crash recovery", which required the data system to process most of a day's ASX data so that it would be ready to answer queries. This was taking several hours, rather than a few minutes, which was clearly unacceptable – a failure of this system would mean that the system would be down for most of the day.

After some analysis it was determined that the bottleneck was in the MS SQLserver database server. The HSBC IT team was rather keen on keeping this component (we offered to switch it to an open source product!) so we spent several days with them trying to improve the performance. Our efforts yielded a factor of two improvement, but this was nowhere near enough, the system was still going to take many hours to recover from an intra-day failure. With just a few days before we needed to freeze the code we finally got approval to retarget the database to Linux/MySQL to see if this will solve the performance issues. In an emphatic rebuttal to the "you get what you pay for" maxim, the MySQL system on identical hardware blew the performance problems away. The SQLserver system would start a recovery simulation processing around 300 messages per second, it quickly deteriorated to somewhere around 70-90 messages per second. At the time, the ASX was generating a peak rate of around 90 messages per second (with a typical sustained rate of 20-30 messages per second) so the system quickly got to the point where it would only slowly catch up to the ASX data.

The MySQL server produced and initial rate of 800 messages per second and could manage a sustained rate of over 300 messages per second throughout the recovery simulation. This meant that the system would be able to recover from a crash and be ready to process live requests within 15 minutes. The other side effect of this success is that it meant that all of the new systems were now open source based.

Performance

The system went live on August 12 1999 after a few weeks of internal use, load and acceptance testing. Over the last 3 years the system has exhibited excellent reliability, due in part to the inherent redundancy (there have been a number of single point hardware failures). Where there have been outages, they have all been related the internet connectivity of the system, not the servers or the software systems.

The system has won several awards including best Online Broker in 2001 and always features at the top of the trade execution speed tests that measure the time taken from trade submission until completion. In terms of maintainability, the system has performed well, and is administered by two HSBC IT staff, they have extended the system beyond equities to other financial vehicles, foreign equities and to provide the interface to online banking. The simplicity of adding new commands to the message switching layer has allowed it to be used to connect additional services.

The addition of the St George customer base later in 1999 proved that the system could deal with large loads without the need for redesign. The underlying software systems only required a few lines of code changed to support St George or other "white labels".

Conclusions

The use of open source software in a mission critical environment has been demonstrated to provide high levels of reliability and performance. Despite the very high peak demands of equities trading platforms, careful design can allow commodity desktop systems to be used to build a system with performance at least the equal of systems costing

orders of magnitude more (we have heard that some banks have spent several million dollars on their online equities platforms, total hardware costs for this project were significantly less than $100k, and the systems are running with a load average of .1).

The Python language has proven to be one of the most important factors to the success of the project. It provided the rapid application development environment that allowed for the production of simulators for all of the main functions, giving all development teams the ability to work in parallel and to measure performance early. Python also lived up to its reputation as a highly expressive language, the total code base at the time of launch was about 2700 lines of code across all servers inclusive of CGI, message servers, database servers and chart generation package. Another major advantage of the prototyping techniques advocated by RAD and extreme programming is that the business groups were able to see and use the system from the earliest stages. This enabled them to test the usability of the site and to make refinements at very early stages of development. Access to a demonstration system also proved to be a critical factor to the successful sales process to St George Bank.

Beyond the obvious technical successes that the open source technologies brought to the project, it is unlikely that any project with an aggressive timeline such as this could have succeeded without the commitment and trust of the bank management and IT staff involved in the project. A lot of what was proposed and used in this system was either unheard of or explicitly against existing bank IT policy, it took a lot of work and guts to get our approach past all the levels of scrutiny within the bank.

In 2001, we undertook a similar project for HSBC in Hong Kong using a very similar architecture and the same tools. Although the number of online users was considerably higher (350,000) the system and techniques scaled up to the challenge and the project has had equal success.

System Monitoring – Much More than Red and Green

Michael Selig
Director of Architecture
Functional Software Pty Ltd
michael_selig@fs.com.au
www.fs.com.au
www.sentinel3g.com.au
July 2002

Introduction

Most commercial data centres these days provide services that are critical to the day-to-day functioning of their entire organisation. These services and the infrastructure supporting them (hardware, operating system, database etc) should be monitored to ensure that everything is "healthy".

The simplest form of System Monitoring is for a competent System Administrator to write one or more scripts (for example to ping a remote host, or to test that an application is running), and to run these scripts periodically, typically via CRON. In this paper, I shall refer to these "information gathering" processes as agents. If the agent finds something unexpected (such as the host not responding to the PING or that the application is down), the script notifies someone (perhaps via email or an SMS message) that there is a problem. Many products refer to this as being in either a "green" or "normal" state, or a "red" or "alarm" state.

Although this is undoubtedly very important information, imagine you are a patient in a critical condition in hospital with your life signs being monitored. I think you would expect the monitor to tell the doctors much more than whether you are "dead" or "alive". Most important is the ability to detect that there will be a problem soon, so that something can be done to rectify the situation in time. This leads us to the next level of monitoring – the "orange" or "warning" state. In order to detect this, our agents must now return more information. Typically this is a number, whose value is tested against a series of thresholds to determine which of the three states ("red", "orange" or "green") the object being monitored is in. For example, when monitoring the disk space, the agent may return the "percentage free", and it may determine that below 2% free is "red", between 2% and 10% is "orange" and above 10% is "green". Alternatively the agent may return a set of discrete states. For example a printer may be "idle", "printing", "offline", "out of paper", "jammed" etc. Each of these states would then have to be mapped to one of the 3 colours.

As an aside, the phrase "the representation of the object being monitored" is a bit of a mouthful, and unfortunately there is no industry standard word for it, so we made one up: the "sentry". This name was derived from the name our System Monitoring product: Sentinel3G, and I'll be using it during this paper. Examples might be "the CPU usage of host SNOOPY" or "the amount of free space on disk X".

Having an agent return simply "red", "orange" or "green" means that you lose a lot of potentially valuable information. For example:

- In the "printer" example, both "idle" and "printing" would probably be mapped to "green", but when a user looks at a console window displaying the status of the printer it would be nice to tell if the printer is idle or busy (plus possibly other associated information).
- Most sites want to log data for historical reporting, graphing and analysis.
- A frequent requirement of a monitoring product is to be able to perform one or more responses on entering any given state.
- Another common requirement is to be able to produce Service Level Reports based on logged state information.

- State-specific documentation and reporting is extremely useful, for example to give users some guidelines to look for the solution to a problem based on the current state.

To implement these, having 2 or 3 states is just not sufficient. This means that in any non-trivial System Monitoring product, an agent cannot simply map everything to 2 or 3 states. They must at least return some "raw data" (such as disk space figures, or printer state) also.

Why Build a System Monitoring Product?

There are a number of commercial and free products available, but we found that customers were generally not particularly satisfied with them for a number of reasons:

- The low cost or free products are often not much more than the "red or green" variety. In fact some of the high cost products aren't much better!
- Those few with the features that are required by most of our customers are both too expensive and too complicated to configure, and tend to be heavy on system resources.
- Many are oriented towards network monitoring, rather than generic system monitoring.
- Most lack the flexibility to be able to "plug in" existing scripts written in any language (including Bourne Shell scripts) as agents.

As a result we came to the conclusion that there was a good market for a low-cost, flexible, network-wide System Monitoring product aimed at small to medium sized enterprises with good reporting and response capabilities. To our surprise, once we released it, we found that large organisations wanted it also, citing the expense and complexity of the existing "high-end" products.

Also, although the process was far from easy, we got an Australian Federal Government R&D grant to build Sentinel3G. Luckily, I wasn't the person that had to do the paperwork!

A State-Machine Architecture

Certainly there are ways to implement features such as responses, state-specific documentation and reports on top of a simple 2 or 3-state monitoring architecture. Typically this involves the user defining a set of conditions based on raw data returned by the agents for each response and again for each category in a Service Level Report. This is in addition to defining the mapping of the raw data to the 3 states "red", "orange", and "green".

When designing Sentinel3G we took a step back and asked ourselves whether there was a way of unifying all these conditions and in so doing, make the job of the user configuring the product simpler, and our job of designing and building it quicker.

One key design decision was to separate the concepts of "state" and "severity". The use of the words "red" and "green" is really a simple way to describe how to display a sentry on a console, giving an indication of how "serious" a problem is. In Sentinel3G, we called this the "severity", and in fact we decided to allow 7 severity levels: disabled, normal, information, warning, alarm, severe and critical. We also decided that a sentry could be in one of an arbitrary set of states, and that each state would have an associated severity. We also decided to implement these states using quite an old, well-known Computer Science concept: the finite-state machine.

In a nutshell, each sentry can have one or more states associated with it and conditions define whether and how to move from one state to another, based on data returned from one or more agents. The definition of an event being detected is simply when the data from an agent causes a sentry to change state.

The diagram shows a simple example of the states of a sentry monitoring disk space. The idea is that the state name would convey something meaningful (eg: "printing", "offline" etc for a printer). Notification of problems, execution of responses, reports and documentation to assist users diagnose problems can simply be associated with the appropriate state(s).

You may have noticed that some of the arrows on the diagram are not labelled with a condition. This is because I have assumed in this example that all transitions to the OK state are identical. This is usually true in most practical cases. As a result, rather than defining separate conditions for every possible state transition, we instead define a single condition to enter a state. (We also provide a pre-defined variable PrevState to handle the rare cases when you need to distinguish transitions from different originating states). In order to simplify state conditions further, they are evaluated in order (usually of severity – with the highest being checked first) stopping when a condition evaluates "true". Should no conditions be satisfied, the pre-defined state Undefined is entered. Finally a null condition always evaluates to "true". The table below shows how the above disk space example might be represented.

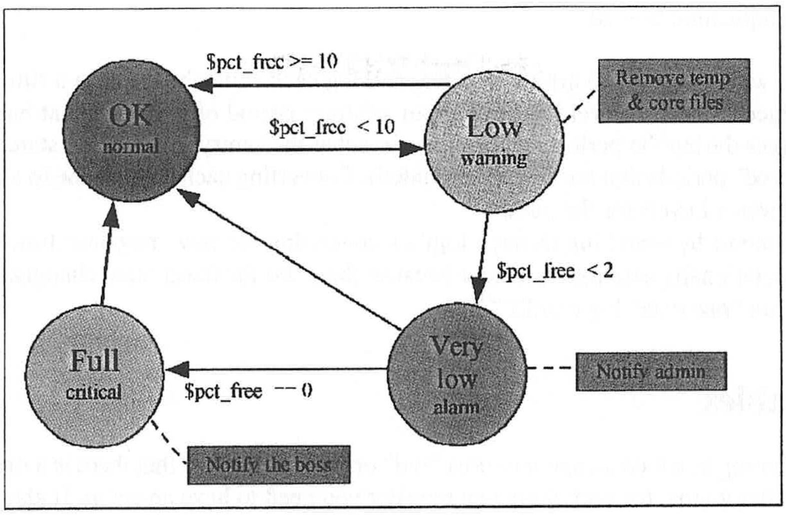


Figure 1: Example of the states of a sentry monitoring disk space

State	Severity	Entry Condition	Delay	Responses
Delete	-	no-data		
Full	critical	$pct_free == 0	0	Notify the boss
Very low	alarm	$pct_free < 2	0	Notify admin
Low	warning	$pct_free < 10	0	Remove temp & core files
			300	Notify admin
OK	normal	-		

Table 1: Representation of disk space example

I have introduced a couple of things in this table that warrant some explanation. Firstly, each state can have a sequence of responses attached to it each with an optional delay before each is performed. For example, "admin" will be notified 5 minutes after entering the Low state. However should there be another state change before this time, the outstanding responses are not performed. In this example, if the response "Remove temp & core files" succeeds in increasing the free space back to over 10%, then no one will be notified.

Secondly, there is a special "no-data" condition which is true if the agent stops returning data for this sentry. If the disk being monitored is unmounted, the agent (which may be running the DF command) no longer reports on this disk, causing the "no-data" condition to become true. The special pre-defined state Delete that has an implied response to delete the sentry, so the effect is that when the disk is unmounted, its icon will disappear from the user's console window.

Service Levels

The use of Service Level Agreements to define a minimum level of service that an organisation either guarantees or aims to achieve is extremely common these days. They are used not only between Service Providers and their customers, but also internally within a single organisation.

A typical Service Level guarantee is made up of a series of "bands", such as: Application X will have a response time of

- Less than 5 seconds for 95% of the time, and
- Less than 10 seconds for 98% of the time

The first thing to be done to measure this is to write an agent (probably a shell or Perl script) which runs a harmless transaction that closely simulates a typical user transaction, and returns its response time. Then create a sentry whose states are:

- Ok: $response_time < 5
- Slow: $response_time < 10

- Excruciating: $response_time >= 10

A monitoring product which can define arbitrary states, and log each state change with a time stamp, makes it extremely simple to produce a Service Level Report over an arbitrary period of time. All that has to be done is to find all logged state changes during the period and sum the time that the sentry was in each state. (Note that there may be some "not monitored" periods that are usually excluded). Converting each state's time to a percentage of the total time gives you the Service Levels for the period.

Implementing such a report by searching through logfiles containing the raw "response_time" data is possible too, but is far more computationally expensive, mainly because there are far fewer state changes during any given period of time than there are "raw data" log entries.

Agents and Variables

The simple-minded monitoring in which an agent returns "red" or "green" implies that there is a one to one mapping of agents to sentries. In other words, for each thing you monitor you need to have an agent. It also usually limits an agent to returning a single piece of information, which can be rather inefficient and also tedious to implement.

Imagine you want to monitor two application programs whose server processes should always be running and they both use the same underlying database. You might consider writing two very similar agents that test if the database is up and run ps (the UNIX process status command) to determine whether their application server process is running. There are 2 inefficiencies here:

1. Both agents test whether the same database is up. It would be simpler and more efficient to have a single agent to do this and for the two "application status" sentries to share the results of this "database status" agent.

2. Both agents independently execute the ps command periodically. It would be much more efficient for a single agent to run the ps command once on behalf of all sentries which need to access process information.

As a result of this, another key design decision was to de-couple agents and sentries. Agents simply become methods to collect data and set variables within the "monitoring engine" which are available to one or more sentries. To continue the example, you might write a "database status" agent that returns a number of variables to do with the database: whether it is up, plus perhaps various performance statistics about the database. The variables from this one agent could then be shared by a number of different sentries including the two "application status" sentries.

An important corollary of this is that states and severities are associated with sentries, not with agents. The variables provide the interface between the agents and sentries.

Though the values of variables are typically numeric (integers or floating point), they may also be strings, boolean or date/time. These variables may be used in state conditions to detect when an event has occurred. They are also used for reporting and graphing, and may be logged for historical reporting.

Another key goal of the design of Sentinel3G was to make the development of new agents as simple as possible. In fact, as well as standard methods of returning data such as SNMP and monitoring log files, we wanted to be able to use standard UNIX tools such as SAR, VMSTAT, DF etc as agents. To do this we had to develop a very flexible "class-based" agent interface. For example the agent interface class which supports the execution of a UNIX program or script can extract the data of interest from the superfluous text around it, saving the user from having to massage the data him or herself using SED, GREP et al.

Although agents are typically "polled" (that is, they are run periodically to return data), we also needed to support external agents which "push" data to Sentinel3G and also ones that run to an arbitrary schedule (eg: weekdays at 5pm).

The Architecture in Brief

Sentinel3G was designed to monitor more than a single host. As we needed to be able to present a unified view of everything being monitored across the enterprise, we used RPC-based communications to a central process – the Event Manager.

The Host Monitor is a per-host daemon process that receives data from its agents, detects events, performs logging and notifies the Event Manager when events occur. It performs discovery (see below) and all responses, including notification, and is responsible for scheduling its agents. The Host Monitor is the main "engine" of Sentinel3G. Note that agents typically execute on the same host as their parent Host Monitor, so usually no network bandwidth is consumed sending data from agent to Host Monitor. Each Host Monitor runs independently, ensuring that its host will be monitored regardless of whether other hosts (including the Event Manager host) or networks are down.

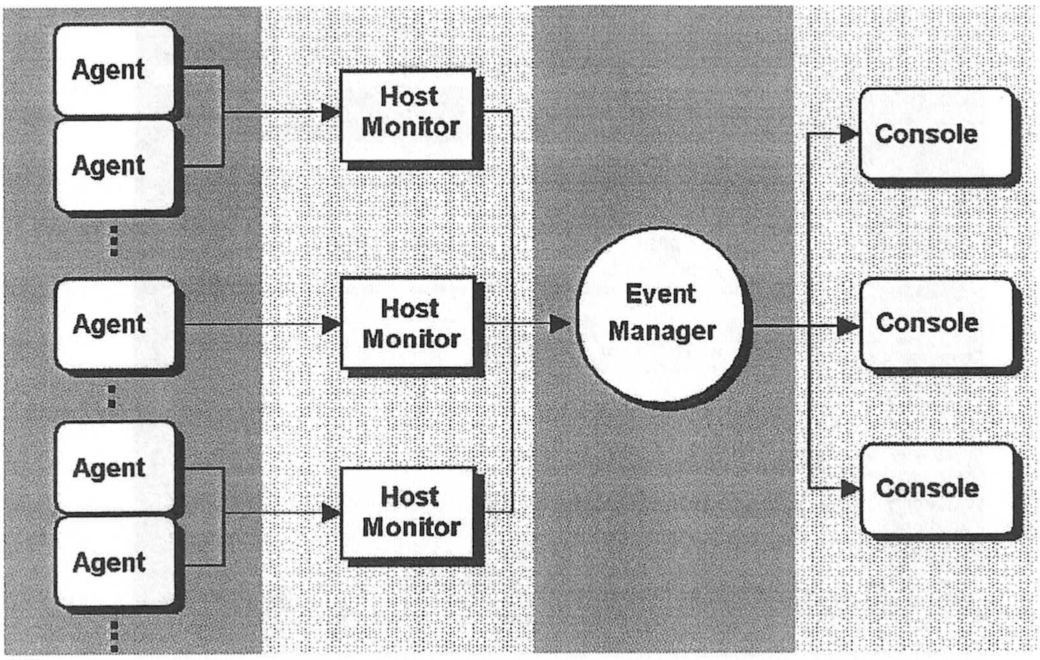


Figure 2: Sentinel3G architecture

The Event Manager is the central process that collects state information from all Host Monitors, performs logging, and updates the data on the Consoles when required. Only state change and updated console information is sent across the network from each Host Monitor to the Event Manager, thus keeping network traffic to a minimum. The Event Manager can run on one of the hosts being monitored, or on a dedicated UNIX or Linux workstation.

The Console is the primary user interface to Sentinel3G, providing one or more different hierarchical views of the sentries being monitored. Each sentry is displayed as an icon and text. The Console also allows arbitrary hierarchical grouping of sentries in folders. When an event occurs, the corresponding sentry changes state, for example, from "up" to "down". This fact is propagated up the hierarchy of folders, so that the user is alerted when something is wrong, even though the actual sentry concerned may not currently be displayed. The user can "drill-down" the hierarchy to get more details, or display different "views", for example, to display only abnormal sentries of a particular type. Access security may also be assigned to folders providing tailored views of certain sentries to certain groups of users.

The flow of data though the system is as follows:

- An agent returns some data to the Host Monitor, setting some variables;
- The state conditions of all sentries using the agent are then evaluated in order. The first to evaluate to "true" causes a change to that state;
- If a state change occurs:
  - the Event Manager (if it is up) is informed;
  - the state change is logged;
  - any pending responses for the previous state are cancelled;
  - if the new state has associated responses, they are scheduled;
- Any changes to the console text, icons etc are sent to the Event Manager (if it is up), and from there to the active consoles;
- Any logging of variables is then done.

Another key design decision was that the product had to be robust. From the architectural block diagram above, it appears that the Event Manager may be a "single point of failure". However, in fact it is a non-critical process: monitoring, logging and notification will all continue in each Host Monitor even if the Event Manager fails. What will be lost in this case are the consoles, though most of our customers find about events via notification – not by sitting in front of a screen. (Having said that, we are working on a fail-over Event Manager for a future release).

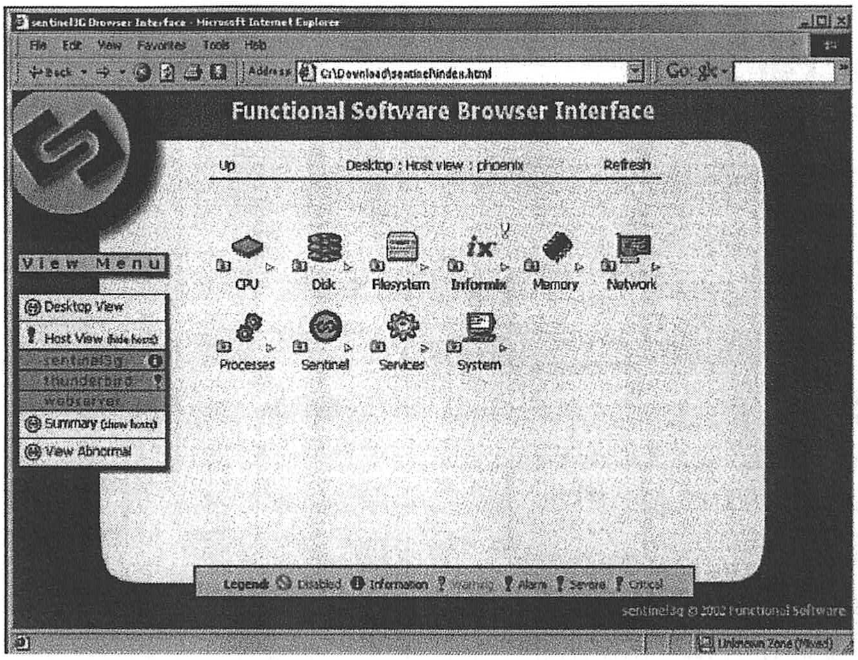


Figure 3: Sentinel3G browser interface

Discovery and Instances

A commercial System Monitoring product must have a range of monitors that work "out of the box". This quite straightforward for simple things like detecting when swap space is low. Things become a little trickier when it comes to monitoring disk drive performance, filesystem free space or a database system, because there may be several of each on any system, and there will have different names and/or locations on each particular installation. We refer to these as "instances".

To make matters more complicated, sometimes a single agent is able to return all the data for all instances of a certain class (eg: DF returns all the data for all mounted filesystems). In other cases, you may have to use separate agents for each instance (for example to monitor each database on a server). In such a multi-instance situation, variables returned are no longer a single scalar set, but become arrays indexed by the instance name.

In either case, sentries must be created for each instance being monitored. We call the process of finding such instances and creating copies of sentries and possibly agents "discovery".

When an agent returns data for all instances (as in the DF example), new instances are immediately discovered and conversely so are ones that disappear. This allows sentries to be dynamically created and destroyed when, for example, and removable disk is mounted or unmounted.

In the case when each instance requires its own agent(s), discovery is achieved by executing a separate program or script. This program builds a table of instances (and possibly associated information) from which agents and sentries are created for each.

Implementation

In order to keep Sentinel3G a low cost product, we needed to build it fairly rapidly with minimal resources. We had a team of only 4 developers, and a part-time tester.

Because our design requires that users be able to manipulate numeric and string variables and specify arbitrary boolean conditions, we decided to use an interpretive language rather than writing our own language & parsers. The Tcl language from Sun Microsystems with incr Tcl object-oriented extensions from Lucent Technologies Inc was chosen to implement the bulk of the product.

The rationale for choosing Tcl over other languages such as Java or C++ is:

- A subset of Tcl syntax was used in Sentinel3G's own configuration (state conditions, variable expressions etc). Tcl is able to directly parse and execute these at runtime.
- Tcl can be used as an extension language both by end-users to develop agents.
- Tcl has a number of neat extensions such as scotty, for network & SNMP support.

- Tcl is a mature, stable Open Source product, and a number of different groups are working on it and extensions.

Although Sentinel3G is not Open Source, all the Tcl source code is provided, which makes it feasible for users to add new agent interfaces and data manipulation functions, for example.

Naturally Sentinel 3G had to be efficient. At first we were a little concerned about how well a monitoring product primarily written in an interpretive language would perform. As a result we ran a number of benchmarks and found that the overhead of using Tcl was surprising low. Tcl's ability to link in compiled C code functions where necessary has enabled our code to be optimised for performance. In fact we found that the biggest performance win is to keep the number of agent executions to a minimum, by

- Keeping poll times reasonable,
- Returning as much useful data as possible in each agent, and
- Using shared agents where possible (eg: for process information)

plus of course, to make the agents execute as efficiently as possible.

One of the main goals of the design was to make Sentinel3G simple to configure. We wanted to avoid the large, complex ASCII configuration file(s) often associated with such products. Luckily we had a head-start here: a toolset that had been developed (mainly in C) for use with previous products that provided a database table interface using ASCII files, and a GUI (itself primarily built using Tk, the GUI extension for Tcl) to manipulate these tables. This made the task of building a completely GUI-based configuration easy.

Remote Monitoring

We are working on a number of new features for Sentinel3G, and I'd like to briefly discuss one of the most important and exciting: Remote Monitoring.

Since releasing Sentinel3G, we have been approached by a number of Service Providers who have asked us how they can provide remote monitoring to their customers. Sometimes the customer's machines are being completely administered by the Service Provider, sometimes even at the Service Provider's premises. Usually, however, the Service Provider is only providing part of the total IT services being used by a client. Furthermore, the client may be also using other Service Providers and may have their own in-house services, all of which need monitoring.

So from the client's perspective:

- They would like to use a single monitoring product (hopefully ours!) to monitor all their services, whether supported in-house, or by external Service Providers.
- They may also want to allocate the monitoring of certain services to those external Service Providers responsible for providing those services.
- Security is important: Service Providers should only be able to access what has been allocated to them.

From the Service Provider's perspective:

- They have lots of clients whom they would like to provide the monitor service.
- The clients may be running different monitoring products.
- There are network firewalls to cross.
- There are potentially thousands of individual sentries to monitor across all clients.

The solution we are working on revolves around a "super" Event Manager – a process running on the Service Provider's host, talking to all the client's Event Managers typically via the open Internet, providing a view of all the sentries relevant to their service across all their customers. The existing Event Manager process running at each customer will usually require a proxy server to handle firewall issues. This proxy will also provide interfaces to events generated by third party monitoring products, providing a uniform external interface. Network security will be enhanced to use Open SSL, and we shall provide an authentication mechanism such as a certificate issued by the customer to the Service Provider so that the Service Providers can be authorised and identified by the customer's Event Manager.

We are currently looking at using the HTTPS protocol to implement this. In brief, each customer's Event Manager would make the sentry information for each Service Provider available at a URL. Because of the large numbers of sentries being monitored by the Service Provider, scalability is a major issue. We need to ensure that state changes are promptly forwarded to the Service Provider, and that the notification and display methods at the Service Provider can cope with thousands of sentries, of which potentially hundreds may be "abnormal". The "folder" implementation of the current Event Manager and Console should be adequate, though we may have to optimise some of the algorithms.

Conclusion

With a small team we managed to build a fully featured, very flexible System Monitoring product in less than two years that is now being successfully used in many commercial and government organisations. We were able to achieve this by using existing technologies where appropriate which at a guess shortened development time by a factor of 5 to10. During the design we also came up with a number of smart and unique approaches to monitoring.

I hope that by using our product as an example I have managed to demonstrate that System Monitoring involves far more than "red" and "green". There are many issues to consider when looking at System Monitoring products, including:

- Ease of use;
- Graphing & reporting capabilities;
- Automatic and manually initiated responses;
- Flexible user notification;
- Ease of installation, configuration and flexibility to build new agents;
- Robustness, efficiency and impact on the rest of the system.

[Marketing hat on] This paper covers only some of the more interesting features of Sentinel3G product. If you would like more technical details about the product, there is a White Paper available at www.sentinel3g.com. Also at this site is a full-function version for Linux available free to individuals and small non-profit organisations.

Duplicate Packets in an IP Trace

Ian D. Graham, Jörg B. Micheel, Gerard K. Sharp, Shao-Lin Joseph Chung
{ian,joerg,gks1,sc57}@cs.waikato.ac.nz
Department of Computer Science
The University of Waikato
Hamilton, New Zealand
Phone: +64-7-8384136
Fax: +64-7-8384155

Abstract

In order to make passive measurements of delay and loss in computer networks it is important to unambiguously distinguish packets from one another. In this paper we present a study of packet replication using a trace file taken at the University of Auckland Internet access link.

Introduction

In order to make passive network measurements of delay and loss in computer networks it is necessary to follow the path of packets through the network. This can be done by a process of packet matching, which requires the unambiguous identification of packets from one another.

In a project to investigate suitable packet signatures for hardware-assisted packet matching, one of the authors (gks) discovered that the IP trace under study contains many duplicated packets; in fact, one packet occurred 409 times [2].

The trace used for this study is a 22 hours recording of the bi-directional traffic on the ATM access link that connects the University of Auckland to the Internet [1]. The entire trace consists of about 50 million packets, from which 783,459 packets, or about 1.56%, are duplicates (Table1). Interestingly, there are about 80% more duplicates originating from the university then entering it. This finding of 1-2% duplicates correlates well with another four day contiguous study at the same point, but collecting IP packet headers: Auckland-VI [6].

Packet duplications may occur for a number of reasons. Duplications can occur either in end systems (originating host) or intermediate devices (switches, routers).

If a source host intends to send identical messages, the only discriminator between them may be the IP ID field. But, the IP specification makes explicit allowance for the higher layer protocol (TCP, UDP) to specify the identification number, in order to improve the chances for successful reassembly of fragmented packets. Another reason for duplication is the limited identifier space of 16 bits. With today's high-speed network links, an IP sequence number wrap may occur in much less than one second.

If intermediate network devices duplicate a packet, it is typically due to some hard- or software fault.

Network anomalies of this kind have been observed previously. Paxson studies packet duplicates in [4], and also discusses the details of TCP/IP protocol implementations and their irregularities. Another related study focuses on

Attribute	From Internet	To Internet
File size (bytes)	17,938,206,72	14,217,496,192
ATM cells	280,284,480	222,992,128
VC cells	280,005,361	222,768,623
IP packets	24,095,382	26,013,496
Duplicated packets	279,848	503,611

Table 1: Packet trace summary

Count	Type	Content	Packet size
409	ICMP	Echo reply	28
107	ICMP	Echo request	28
33	ICMP	TTL exceeded	56
17	TCP	ACK	40
16	TCP	ACK	40
14	NBMA	NHRP Res. Req.	68
14	NBMA	NHRP Res. Req.	68
12	NBMA	NHRP Res. Req.	68
12	ICMP	Echo request	28
11	ICMP	Echo reply	1500

Table 2: Packet types for most frequent duplicates

IP checksum errors and their causes with respect to hardware malfunction [5].

Terminology

With identical packets appearing multiple times it is important to understand how to count them. If a single packet appears exactly twice, we consider this as one non-unique packet having two duplications. (A packet with one duplication thus is actually a unique packet.)

Analysis Method

In order to distinguish unique and non-unique packets we represent packets by identifiers. We convince ourselves that indeed there is a one-to-one relationship between packet identifiers and the packets [2]. We find that MD5 checksums, while very robust against accidental or malicious false matches, are very difficult to compute in hard- and software. Instead, we chose a CRC64 checksum, which, like the well-known CRC32 checksum used in Ethernet, is relatively easy to compute. A signature is computed for all packets, these are sorted and uniqued, then packets thus shown to be identical are confirmed via byte-by-byte comparison.

Investigation of non-unique Packets

We start our investigation of the most frequent duplications by looking at the types of packets:

As we can see from Table 2, most of the frequently replicated packets are small and do not carry user data payload, but are control packets of different kinds. However, this analysis does not cover the bulk of the replicated packets.

We further investigate duplicate packets by source and destination hosts. We found that one particular pair of hosts with four incarnations of an FTP session accounts for almost 97.6% of all duplicated packets. The TCP log documents erratic protocol behaviour at one of the end systems, whereby a host sends a data packet with sequence number N, then within a second sends another acknowledgement packet with no data using the same sequence number N, which triggers a burst of retransmissions from the remote end. At this point, both hosts appear to get out of sync and start retransmitting with long bursts of acknowledgement packets. We later eliminated those sessions from further analysis of replicated packets.

From the point of view of network delay and loss studies it is useful to understand the time interval between incarnations of the same duplicate, and their distribution. To state it simply, if packets are minutes or hours apart, they are unlikely to produce false matches for delay or loss analysis, as packets are unlikely to survive for this long inside the Internet, due to the IP TTL mechanism.

Without eliminating the offending pair of hosts from the analysis, only about 1.5% of all intervals between duplicates are within 29 seconds apart from each other (Figure 1). However, after eliminating the offenders, about 86% of all packets within the 29-second interval.

In fact, 75% of all duplicates are now within a one second interval from each other. Still, the total number of duplicates is now reduced to about 0.04% of all packets (Figure 2).

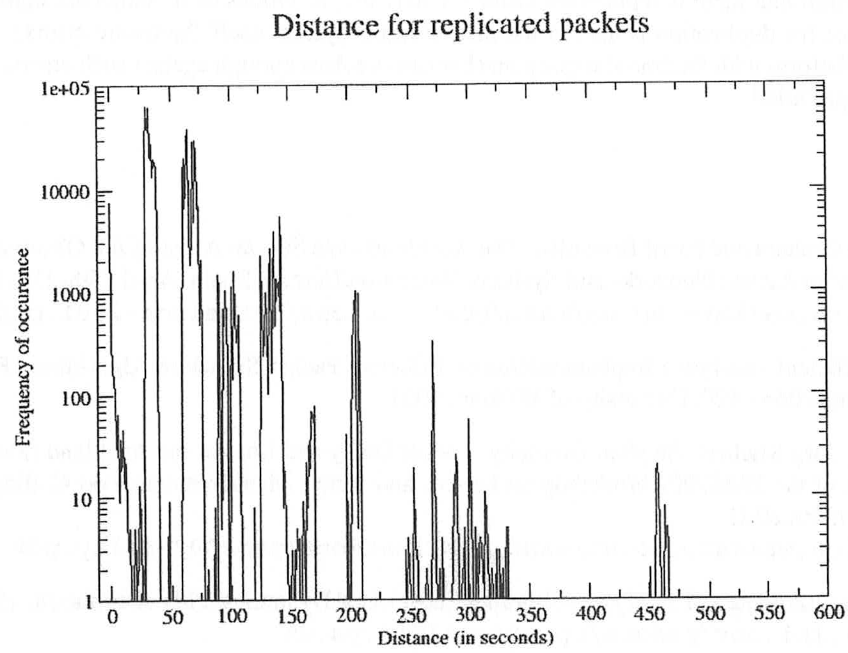


Figure 1: Distribution of time distance for duplicates

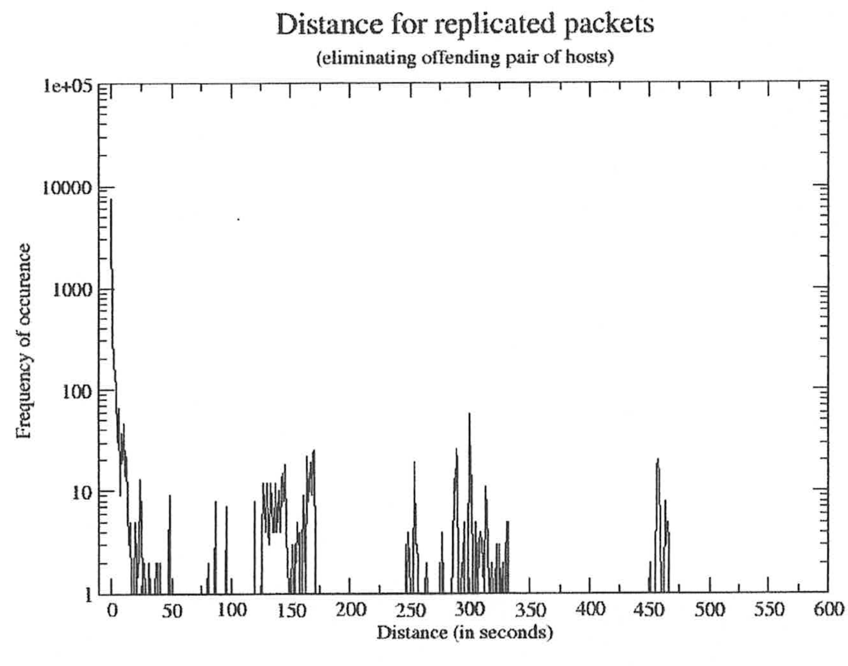


Figure 2: Same distribution, but eliminating offenders

Discussion

Duplicated packets are very interesting. This study is limited to one observation point only. Looking at source IP addresses leads to suspicion that there is replication inside the network, as blocks of IP addresses share a common prefix. A possible source for duplication is also in the measurement system itself (hardware errors). We believe that the Dag hardware platform with its time stamping mechanism is robust enough against such errors, however no 100% guarantee can be provided.

References

1. Jörg Micheel, Ian Graham and Nevil Brownlee: The Auckland Data Set: an Access Link Observed, 14th ITC Specialist Seminar on Access Networks and Systems, Barcelona/Gerona, Spain, April 25th-27th 2001. `http://wand.cs.waikato.ac.nz/wand/publications/barcelona-2001.pdf`

2. Gerard Sharp: Efficient Hardware Implementation of Effective Packet Signature Algorithms, Report of an Investigation, Course 0657.420, University of Waikato, 2001.

3. Klaus Mochalski, Jörg Micheel, Stephen Donnelly: Packet Delay and Loss at the Auckland Internet Access Path, Proceedings of the PAM2002 Workshop on Passive and Active Measurements, Fort Collins, Colorado, USA, March 25th/26th 2002. `http://wand.cs.waikato.ac.nz/wand/publications/pam2002-delay.pdf`

4. Vern Paxson: Measurements and Analysis of Internet End-to-End Dynamics, PhD dissertation, 1997. `ftp://ftp.ee.lbl.gov/papers/vp-thesis/dis.ps.gz`

5. Jonathan Stone and Craig Partridge: When the CRC and TCP checksum disagree, ACM SIGCOMM2000, August 2000, Stockholm, Sweden. `http://www.acm.org/sigs/sigcomm/sigcomm2000/conf/paper/sigcomm2000-9-1.ps.gz`

6. WAND WITS Auckland-VI illustrated IP header trace file, `http://wand.cs.waikato.ac.nz/wand/wits/auck/6/`

The Good, the Bad, and the Ugly: The Unix Legacy

Rob Pike
Bell Labs
Lucent Technologies
Room 2C526
600 Mountain Ave
Murray Hill, NJ 07974
USA
`rob@plan9.bell-labs.com`

Abstract

In its billion second lifetime, the Unix system has had a profound effect on the world of computing. It has fostered new programming languages and models, been a key player in the development and quotidian running of the Internet, and influenced just about every major computing environment that followed it. Even the lowly URL looks like a Unix file name.

But all is not well with Unix. Its very success has led to problems: multiple versions; antiquated standards; an old-fashioned, famously unfriendly user interface.

In this talk, I will argue that Unix's strengths are also its weaknesses: that what makes the system good at what it's good at is also what makes it bad at what it's bad at. Moreover, recent attempts to address the weaknesses have tended to diminish its strengths. The problems can be overcome, but they require some re-evaluation of the direction the system's evolution is headed. One thing they do not require is another version of Unix.

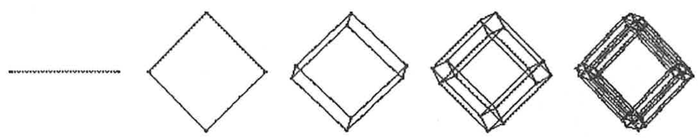
Thursday 5th September 2002

Gnutella, GRIDs and Gargantuan Computing

Dr. Neil J. Gunther
Performance Dynamics Company^{SM}
Castro Valley, California, USA
`www.perfdynamics.com`

The presentation will go into more detail than this abstract about why these networks:

(which represent typical P2P topologies) should be replaced by hypernets like these...



to guarantee the kind of scalability that has been promised by various gargantuan computing architectures.

Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux

Hubertus Franke
IBM Thomas J. Watson Research Center
Rusty Russell
IBM Linux Technology Center
Matthew Kirkwood
frankeh@watson.ibm.com, rusty@rustcorp.com.au, matthew@hairy.beasts.org

Abstract

Fast userlevel locking is an alternative locking mechanism to the typically heavy weight kernel approaches such as fcntl locking and System V semaphores. Here, multiple processes communicate locking state through shared memory regions and atomic operations. Kernel involvement is only necessary when there is contention on a lock, in order to perform queueing and scheduling functions. In this paper we discuss the issues related to user level locking by following the history of ideas and the code to the current day. We present the efficacy of "futexes" through benchmarks, both synthetic and through adaptations to existing databases. We conclude by presenting the potential future directions of the "futex" interface.

1. Introduction

Linux(TM)[1] has seen significant growth as a server operating system and has been successfully deployed in enterprise environments for Web, file and print serving. With the deployment of Version 2.4, Linux has seen a tremendous boost in scalability and robustness that makes it now feasible to deploy even more demanding enterprise applications such as high end databases, business intelligence software and application servers. As a result, whole enterprise business suites and middlewares such as SAP(TM), Websphere(TM), Oracle, DB2(TM)[2], etc., are now available for Linux.

For these enterprise applications to run efficiently on Linux, or on any other operating system for that matter, the OS must provide the proper abstractions and services. Enterprise applications and applications suites are increasingly built as multi process / multi-threaded applications. Multi-threaded applications can take better advantage of SMP hardware, while multiple processes allows for higher degrees of fault tolerance, i.e., a single process abort does not necessarily bring the entire application down. Furthermore, applications suites are often a collection of multiple independent subsystems.

Despite their functional separation, the processes representing these subsystems often must communicate with each other and share state amongst each other. Examples of this are database systems, which typically maintain shared I/O buffers in user space. The buffers are concurrently accessed by various database engines and prefetching processes.

Access to such shared state must be properly synchronized through either exclusive or shared locks. Exclusive locks allow only one party access to the protected entity, while shared locks allow multiple reader - single writer semantics. Synchronization implies a shared state, indicating that a particular resource is available or busy, and a means to wait for its availability. The latter one can either be accomplished through busy-waiting or through a explicit / implicit call to the scheduler.

In traditional UNIX(TM)[3] systems, System V IPC (inter process communication) such as *semaphores*, *msgqueues*, *sockets* and the file locking mechanism (`flock()`) are the basic mechanisms for two processes to synchronize. These mechanisms expose an opaque handle to a kernel object that naturally provides the shared state and atomic operations in the kernel. Services must be requested through system calls (e.g., `semop()`). The drawback of this

[1] Linux is a trademark of Linus Torvalds
[2] All third party trademarks are the property of their respective owners.
[3] UNIX is a trademark of The Open Group

approach is that every lock access requires a system call. When locks have low contention rates, the system call can constitute a significant overhead.

One solution to this problem is to deploy user level locking, which avoids some of the overhead associated with purely kernel-based locking mechanisms. It relies on a user level lock located in a shared memory region and modified through atomic operations to indicate the lock status. Only the contended case requires kernel intervention. The exact behavior and the obtainable performance are directly affected by how and when the kernel services are invoked. The idea described here is not new. Some of the foundation of this paper are described in [4], [7] and [6]. In [2] the impact of locking on JVM performance is discussed.

In this paper we are describing a particular fast user level locking mechanism called *futexes* that was developed in the context of the Linux operating system. It consists of two parts, the user library and a kernel service that has been integrated into the Linux kernel distribution version 2.5.7.

Requirements

In this section we are stating some of the requirements of a fast userlevel locking mechanism that we derived as part of this work and that were posted to us as requirements by middleware providers.

There are various behavioral requirements that need to be considered. Most center around the fairness of the locking scheme and the lock release policy. In a **fair** locking scheme the lock is granted in the order it was requested, i.e., it is handed over to the longest waiting task. This can have negative impact on throughput due to the increased number of context switches. At the same time it can lead to the so called **convoy problem**. Since, the locks are granted in the order of request arrival, they all proceed at the speed of the slowest process, slowing down all waiting processes. A common solution to the convoy problem has been to mark the lock available upon release, wake all waiting processes and have them recontend for the lock. This is referred to as **random fairness**, although higher priority tasks will usually have an advantage over lower priority ones. However, this also leads to the well known **thundering herd problem**. Despite this, it can work quite well on uni-processor systems if the first task to wake releases the lock before being preempted or scheduled, allowing the second herd member to obtain the lock, etc. It works less spectacularly on SMP. To avoid this problem, one should only wake up one waiting task upon lock release. Marking the lock available as part of releasing it, gives the releasing task the opportunity to reacquire the lock immediately again, if so desired, and avoid unnecessary context switches and the convoy problem. Some refer to these as **greedy**, as the running task has the highest probability of reacquiring the lock if the lock is hot. However, this can lead to starvation. Hence, the basic mechanisms must enable both fair locking, random locking and greedy or convoy avoidance locking (short ca-locking). Another requirement is to enable spin locking, i.e., have an application spin for the availablilty of the lock for some user specified time (or until granted) before giving up and resolving to block in the kernel for its availability. Hence an application has the choice to either (a) block waiting to be notified for the lock to be released, or (b) yield the processor until the thread is rescheduled and then the lock is tried to be acquired again, or (c) spin consuming CPU cycles until the lock is released.

With respect to performance, there are basically two overriding goals:

- avoid system calls if possible, as system calls typically consume several hundred instructions.
- avoid unnecessary context switches: context switches lead to overhead associated with TLB invalidations etc.

Hence, in fast userlevel locking, we first distinguish between the uncontended and the contended case. The uncontended case should be efficient and should avoid system calls by all means. In the contended case we are willing to perform a system call to block in the kernel.

Avoiding system calls in the uncontended case requires a shared state in user space accessible to all participating processes/task. This shared state, referred to as the *user lock*, indicates the status of the lock, i.e., whether the lock is held or not and whether there are waiting tasks or not. This is in contrast to the System V IPC mechanisms which merely exports a handle to the user, and performs all operations in the kernel.

The user lock is located in a shared memory region that was create via `shmat()` or `mmap()`. As a result, it can be located at different virtual addresses in different address spaces. In the uncontended case, the application atomically changes the lock status word without entering into the kernel. Hence, atomic operations such as `atomic_inc()`, `atomic_dec`, `cmpxchg()`, and `test_and_set()` are neccessary in user space. In the contended case, the application needs to wait for the release of the lock or needs to wake up a waiting task in the case of an unlock operation. In order to wait in the kernel, a *kernel object* is required, that has *waiting queues* associated with it. The waiting queues provide the queueing and scheduling interactions. Of course, the aforementioned IPC mechanisms can be used for this purpose. However, these objects still imply a heavy weight object that requires a priori allocation and often does not precisely provide the required functionality. Another alternative that is commonly deployed are *spinlocks* where the task spins on the availability of the user lock until granted. To avoid too many cpu cycles being wasted, the task yields the processor occasionally.

It is desirable to have the user lock be handle-free. In other words instead of handling an oqaque *kernel handle*, requiring initialization and cross process global handles, it is desirable to address locks directly through their virtual address. As a consequence, kernel objects can be allocated dynamically and on demand, rather than apriori.

A lock, though addressed by a virtual address, can be identified conceptually through its *global lock identity*, which we define by the memory object backing the virtual address and the offset within that object. We notate this by the tuple [B,O]. Three fundamental memory types can be distinguished that represent B: (a) anonymous memory, (b) shared memory segment, and (c) memory mapped files. While (b) and (c) can be used between multiple processes, (a) can only be used between threads of the same process. Utilizing the virtual address of the lock as the kernel handle also provides for an integrated access mechanism that ties the virtual address automatically with its kernel object.

Despite the atomic manipulation of the user level lock word, race conditions can still exists as the sequence of lock word manipulation and system calls is not atomic. This has to be resolved properly within the kernel to avoid deadlock and inproper functioning.

Another requirement is that fast user level locking should be simple enough to provide the basic foundation to efficiently enable more complicated synchronization constructs, e.g. semaphores, rwlocks, blocking locks, or spin versions of these, pthread mutexes, DB latches. It should also allow for a clean separation of the blocking requirements towards the kernel, so that the blocking only has to be implemented with a small set of different constructs. This allows for extending the use of the basic primitives without kernel modifications. Of interest is the implementation of mutex, semaphores and multiple reader/single writer locks.

Finally, a solution needs to be found that enables the recovery of "dead" locks. We define unrecoverable locks as those that have been acquired by a process and the process terminates without releasing the lock. There are no convenient means for the kernel or for the other processes to determine which locks are currently held by a particular process, as lock acquisition can be achieved through user memory manipulation. Registering the process's "pid" after lock acquisition is not enough as both operations are not atomic. If the process dies before it can register its pid or if it cleared its pid and before being able the release the lock, the lock is unrecoverable. A protocol based solution to this problem is described in [1]. We have not addressed this problem in our prototypes yet.

Linux Fast User level Locking: History and Implementations

Having stated the requirements in the previous section, we now proceed to describe the basic general implementation issues. For the purpose of this discussion we define a general opaque datatype `ulock_t` to represent the userlevel lock. At a minimum it requires a status word.

```
typedef struct ulock_t {
        long status;
} ulock_t;
```

We assume that a shared memory region has been allocated either through `shmat()` or through `mmap()` and that any user locks are allocated into this region. Again note, that the addresses of the same lock need not be the same across all participating address spaces. The basic semaphore functions `UP()` and `DOWN()` can be implemented as follows.

```
static inline int usema_down(ulock_t *ulock)
{
        if (!__ulock_down(ulock))
                return 0;
        return sys_ulock_wait(ulock);
}

static inline int usema_up(ulock_t *ulock)
{
        if (!__ulock_up(ulock))
                return 0;
        return sys_ulock_wakeup(ulock);
}
```

The `__ulock_down()` and `__ulock_up()` provide the atomic increment and decrement operations on the lock status word. A non positive count (status) indicates that the lock is not available. In addition, a negative count *could* indicate the number of waiting tasks in the kernel. If a contention is detected, i.e. `(ulock->status <= 0)`, the

kernel is invoked through the `sys_*` functions to either wait on the wait queue associated with `ulock` or release a blocking task from said waitqueue.

All counting is performed on the lock word and race conditions resulting from the non-atomicity of the lock word must be resolved in the kernel. Due to such race conditions, a lock can receive a wakeup before the waiting process had a chance to enqueue itself into the kernel wait queue. We describe below how various implementation resolved this race condition as part of the kernel service.

One early design suggested was the explicit allocation of a kernel object and the export of the kernel object address as the handle. The kernel object was comprised of a wait queue and a unique security signature. On every wait or wakeup call, the signature would be verified to ensure that the handle passed indeed was referring to a valid kernel object. The disadvantages of this approach have been mentioned, namely that a handle needs to be stored in `ulock_t` and that explicit allocation and deallocation of the kernel object are required. Furthermore, security is limited to the length of the key and hypothetically could be guessed.

Another prototype implementation, known as *ulocks* [3], implements general user semaphores with both fair and convoy avoidance wakeup policy. Mutual exclusive locks are regarded as a subset of the user semaphores. The prototype also provides multiple reader/single writer locks (rwlocks). The user lock object `ulock_t` consists of a lock word and an integer indicating the required number of kernel wait queues. User semaphores and exclusive locks are implemented with one kernel wait queue and multiple reader/single writer locks are implemented with two kernel wait queues.

This implementation separates the lock word from the kernel wait queues and other kernel objects, i.e., the lock word is never accessed from the kernel on the time critical wait and wakeup code path. Hence the state of the lock and the number of waiting tasks in the kernel is all recorded in the lock word. For exclusive locks, standard counting as described in the general `ulock_t` discussion, is implemented. As with general semaphores, a positive number indicates the number of times the semaphore can be acquired, "0" and less indicates that the lock is busy, while the absolute of a negative number indicates the number of waiting tasks in the kernel.

The "premature" wakeup call is handled by implementing the kernel internal waitqueues using kernel semaphores (`struct semaphore`) which are initialized with a value 0. A premature wakeup call, i.e. no pending waiter yet, simply increases the kernel semaphore's count to 1. Once the pending wait arrives it simply decrements the count back to 0 and exits the system call without waiting in the kernel. All the wait queues (kernel semaphores) associated with a user lock are encapsulated in a single kernel object.

In the rwlocks case, the lock word is split into three fields: write locked (1 bit), writes waiting (15 bits), readers (16 bits). If write locked, the `readers` indicate the number of tasks waiting to read the lock, if not write locked, it indicates the numbers of tasks that have acquired read access to the lock. Writers are blocking on a first kernel wait queue, while readers are blocking on a second kernel wait queue associated with a ulock. To wakeup multiple pending read requests, the number of task to be woken up is passed through the system call interface.

To implement rwlocks and ca-locks, atomic compare and exchange support is required. Unfortunately on older 386 platforms that is not the case.

The kernel routines must identify the kernel object that is associated with the user lock. Since the lock can be placed at different virtual addresses in different processes, a lookup has to be performed. In the common fast lookup, the virtual address of the user lock and the address space are hashed to a kernel object. If no hash entry exists, the proper global identity $[B, O]$ of the lock must be established. For this we first scan the calling process's vma list for the vma containing the lock word and its offset. The global identity is then looked up in a second hash table that links global identities with their associated kernel object. If no kernel object exists for this global identity, one is allocated, initialized and added to the hash functions. The `close()` function associated with a shared region holding kernel objects is intercepted, so that kernel objects are deleted and the hash tables are cleaned up, once all attached processes have detached from the shared region.

While this implementation provides for all the requirements, the kernel infrastructure of multiple hash tables and lookups was deemed too heavy. In addition, the requirement for compare and exchange is also seen to be restrictive.

Futexes

With several independent implementations [8, 9, 10] in existence, the time seemed right in early 2002 to attempt to combine the best elements of each to produce the minimum useful subset for insertion into the experimental Linux kernel series.

There are three key points of the original futex implementation which was added to the 2.5.7 kernel:

1. We use a unique identifier for each futex (which can be shared across different address spaces, so may have different virtual addresses in each): this identifier is the "struct page" pointer and the offset within that page. We increment the reference count on the page so it cannot be swapped out while the process is sleeping.

2. The structure indicating which futex the process is sleeping on is placed in a hash table, and is created upon entry to the futex syscalls on the process's kernel stack.

3. The compression of "*f*ast *u*serspace mu*tex*" into "*futex*" gave a simple unique identifier to the section of code and the function names used.

The 2.5.7 Implementation

The initial implementation which was judged a sufficient basis for kernel inclusion used a single two-argument system call, ``sys_futex(struct futex *, int op)''. The first argument was the address of the futex, and the second was the operation, used to furthur demultiplex the system call and insulate the implementation somewhat from the problems of system call number allocation. The latter is especially important as the system call is expand as new operations are required. The two valid op numbers for this implementation were `FUTEX_UP` and `FUTEX_DOWN`.

The algorithm was simple, the file *linux/kernel/futex.c* containing 140 code lines, and 233 in total.

1. The user address was checked for alignment and that it did not overlap a page boundary.
2. The page is pinned: this involves looking up the address in the process's address space to find the appropriate "`struct page *`", and incrementing its reference count so it cannot be swapped out.
3. The "`struct page *`" and offset within the page are added, and that result hashed using the recently introduced fast multiplicative hashing routines [11], to give a hash bucket in the futex hash table.
4. The "op" argument is then examined. If it is `FUTEX_DOWN` then:
   (a) The process is marked *INTERRUPTIBLE*, meaning it is ready to sleep.
   (b) A "`struct futex_q`" is chained to the tail of the hash bucket determined in step 3: the tail is chosen to give FIFO ordering for wakeups. This structures contains a pointer to the process and the "`struct page *`" and offset which identify the futex uniquely.
   (c) The page is mapped into low memory (if it is a high memory page), and an atomic decrement of the futex address is attempted,[4] then unmapped again. If this does not decrement the counter to zero, we check for signals (setting the error to `EINTR` and going to the next step), schedule, and then repeat this step.
   (d) Otherwise, we now have the futex, or have received a signal, so we mark this process *RUNNING,* unlink ourselves from the hash table, and wake the next waiter if there is one, and return `0` or `-EINTR`. We have to wake another process so that it decrements the futex to -1 to indicate that it is waiting (in the case where we have the futex), or to avoid the race where a signal came in just as we were woken up to get the futex (in the case where a signal was received).
5. If the op argument was `FUTEX_UP`:
   (a) Map the page into low memory if it is in a high memory page
   (b) Set the count of the futex to one ("available").
   (c) Unmap the page if it was mapped from high memory
   (d) Search the hash table for the first "`struct futex_q`" associated with this futex, and wake up that process.
6. Otherwise, if the op argument is anything else, set the error to EINVAL.
7. Unpin the page.

While there are several subtleties in this implementation, it gives a second major advantage over System V semaphores: there are no explicit limits on how many futexes you can create, nor can one futex user "starve" other users of futexes. This is because the futex is merely a memory location like any other until the `sys_futex` syscall is entered, and each process can only do one `sys_futex` syscall at a time, so we are limited to pinning one page per process into memory, at worst.

What about Read-Write Locks?

We considered an implementation of "FUTEX_READ_DOWN" et. al, which would be similar to the simple mutual exclusion locks, but before adding these to the kernel, Paul Mackerras suggested a design for creating read/write lock in userspace by using two futexes and a count: *f*ast *u*serspace *r*ead/*w*rite l*ocks*, or *furwocks.* This implementation provides the benchmark for any kernel-based implementation to beat to justify its inclusion as a first-class primitive, which can be done by adding new valid "op" values. A comparision with the integrated approach chosen by ulocks is provided later.

[4] We do not even attempt to decrement the address if it is already negative, to avoid potential wraparound. We do the decrement even if the counter is zero, as "-1" indicates we are sleeping and hence has different semantics than 0.

Problems with the 2.5.7 Implementation

Once the first implementation entered the mainstream experimental kernel, it drew the attention of a much wider audience. In particular those concerned with implementing POSIX™[5] threads, and attention also returned to the fairness issue.

- There is no straightforward way to implement the pthread_cond_timedwait primitive: this operation requires a timeout, but using a timer is difficult as these must not interfere with their use by any other code.
- The pthread_cond_broadcast primitive requires every process sleeping to be woken up, which does not fit well with the 2.5.7 implementation, where a process only exits the kernel when the futex has been successfully obtained or a signal is received.
- For N:M threading, such as the Next Generation Posix Threads project [5] an asynchronous interface for finding out about the futex is required, since a single process (containing multiple threads) might be interested in more than one futex.
- Starvation occurs in the following situtation: a single process which immediately drops and then immediately competes for the lock will regain it before any woken process will.

With these limitations brought to light, we searched for another design which would be flexible enough to cater for these diverse needs. After various implementation attempts and discussions we settled on a variation of *atomic_compare_and_swap* primitive, with the atomicity guaranteed by passing the expected value into the kernel for checking. To do this, two new "op" values replaced the operations above, and the system call was changed to two additional arguments, "int val" and "struct timespec *reltime".

FUTEX_WAIT: Similar to the previous FUTEX_DOWN, except that the looping and manipulation of the counter is left to userspace. This works as follows:

1. Set the process state to *INTERRUPTIBLE*, and place "struct futex_q" into the hash table as before.
2. Map the page into low memory (if in high memory).
3. Read the futex value.
4. Unmap the page (if mapped at step 2).
5. If the value read at step 3 is not equal to the "val" argument provided to the system call, set the return to `EWOULDBLOCK`.
6. Otherwise, sleep for the time indicated by the "reltime" argument, or indefinitely if that is NULL.
   (a) If we timed out, set the return value to `ETIMEDOUT`.
   (b) Otherwise, if there is a signal pending, set the return value to `EINTR`.
7. Try to remove our "`struct futex_q`" from the hash table: if we were already removed, return 0 (success) unconditionally, as this means we were woken up, otherwise return the error code specified above.

FUTEX_WAKE: This is similar to the previous `FUTEX_UP`, except that it does not alter the futex value, it simple wakes one (or more) processes. The number of processes to wake is controlled by the "int val" parameter, and the return value for the system call is the number of processes actually woken and removed from the hash table.

FUTEX_AWAIT: This is proposed as an asynchronous operation to notify the process via a SIGIO-style mechanism when the value changes. The exact method has not yet been settled (see the section "Current and Future Directions").

This new primitive is only slightly slower than the previous one,[6] in that the time between waking the process and that process attempting to claim the lock has increased (as the lock claim is done in userspace on return from the FUTEX_WAKE syscall), and if the process has to attempt the lock multiple times before success, each attempt will be accompanied by a syscall, rather than the syscall claiming the lock itself.

On the other hand, the following can be implemented entirely in the userspace library:

1. All the POSIX style locks, including pthread_cond_broadcast (which requires the "wake all" operation) and pthread_cond_timedwait (which requires the timeout argument). One of the authors (Rusty) has implemented a "non-pthreads" demonstration library which does exactly this.

[5] POSIX is a trademark of the IEEE Inc.
[6] About 1.5% on a low-contention tdbtorture, 3.5% on a high-contention tdbtorture

2. Read-write locks in a single word, on architectures which support cmpxchg-style primitives.

3. FIFO wakeup, where fairness is guaranteed to anyone waiting (see below).

Finally, it is worthwhile pointing out that the kernel implementation requires exactly the same number of lines as the previous implementation: 233.

FIFO Queueing

The naive implementation of "up" does the following:

1. Atomically set the futex to 1 ("available") and record the previous value.

2. If the previous value was negative, invoke sys_futex to wake up a waiter.

Now, there is the potential for another process to claim the futex (without entering the kernel at all) between these two steps: the process woken at step 2 will then fail, and go back to sleep. As long as this does not lead to starvation, this unfairness is usually tolerable, given the performance improvements shown in the performance evaluation later in the document.

There is one particular case where starvation is a real problem which must be avoided. A process which is holding the lock for extended periods and wishes to "give way" if others are waiting cannot simple to "futex_up(); futex_down();", as it will always win the lock back before any other processes.

Hence one of us (Hubertus) added the concept of "`futex_up_fair()`", where the futex is set to an extremely negative number ("passed"), instead of 1 ("available"). This looks like a "contended" case to the fast userspace "futex_down()" path, as it is negative, but indicates to any process after a successful return from the `FUTEX_WAIT` call that the futex has been passed directly, and no further action (other than resetting the value to -1) is required to claim it.

Performance Evaluation

In this section we assess the performance of the current implementation. We start out with a synthetic benchmark and continue with a modified database benchmark.

MicroBenchmark: UlockFlex

Ulockflex is a synthetic benchmark designed to ensure the integrity and measure the performance of locking primitives. In a run, *Ulockflex* allocates a finite set (typically one) of global shared regions (shmat or mmap'ed files) and a specified number of user locks which are assigned to the shared region in a round robin fashion. It then clones a specified number of tasks either as threads or as processes and assigns each task to one particular lock in a round robin fashion. Each cloned task, in a tight loop, computes two random numbers $nlht$ and lht, acquires its assigned lock, does some work of lock hold time lht, releases the lock, does some more work of non-lock hold time $nlht$ and repeats the loop. The mean lock hold time $lht(mean)$ and non-lock hold times $nlht(mean)$ are input parameters. lht and $nlht$ are determined as random numbers over a uniform distribution in the interval [0.5..1.5] of their respective mean. The tool reports total cummulative throughput (as in number of iterations through the loop). It also reports the coefficient of variance of the per task througput. A higher coefficient indicates the potential for starvation. A small coefficient indicates fairness over the period of execution. A data structure associated with each lock is updated after obtaining the lock and verified before releasing the lock, thus allowing for integrity checks.

In the following we evaluate the performance of various user locking primitives that were built on the basics of the futex and the ulock implementations. We consider the basic two wakeup policies for both futexes and ulocks, i.e. fair wakeup and regular wakeup (i.e. convoy avoidance), yielding the 4 cases *futex_fair, futex, ulocks_fair* and *ulocks*. For these cases we also consider a spinning lock acquisition in that the task tries to acquire the lock for 3 $\mu secs$ before giving up and blocking in the kernel, yielding the 4 cases of *futex_fair(spin,3), futex(spin,3), ulocks_fair(spin,3)* and *ulocks(spin,3)*. For reference we also provide the measurements for a locking mechanism build on System V semaphores, i.e., each lock request results in a system call. This variant is denoted as *sysv*, resulting in 9 overall locking primitives being evaluated.

All experiments were performed on a dual Pentium-III 500 MHz, 256MB system. A data point was obtained by running ulockflex for 10 seconds with a minimum of 10 runs or until a 95% confidence interval was achieved.

In the first experiment we determine the basic overhead of the locking mechanims. For this we run with one task, one lock and $nlht == lht == 0$. Note that in this case all user locking mechanisms never have to enter into the kernel. Performance is reported as % efficiency of a run without lock invocations. The *sysv* was 25.1% efficient,

Conf	no-spin		spin	
	futex	ulock	futex	ulock
2 tasks				
(0,10)	-15.5	-0.7	-20.5	-22.9
(5,5)	7.9	4.6	52.4	47.7
(7,3)	15.5	18.7	50.2	66.4
(9,1)	33.2	33.1	40.1	46.5
3 tasks				
(0,10)	-13.7	-15.2	-19.1	-15.9
(5,5)	-5.7	8.9	-10.1	3.8
(7,3)	-33.0	11.0	-28.2	-9.2
(9,1)	-33.7	7.5	-21.7	-0.7
4 tasks				
(0,10)	-15.8	-20.0	-20.4	-17.5
(5,5)	0.6	13.3	-5.3	13.5
(7,3)	-38.6	8.0	-42.5	7.3
(9,1)	-43.6	7.7	-30.6	6.4
100 tasks				
(0,10)	172.3	190.8	151.4	189.5
(5,5)	367.6	393.9	386.4	397.6
(7,3)	464.0	300.5	449.0	305.5
(9,1)	495.7	180.3	449.1	190.0
1000 tasks				
(0,10)	1900.4	2343.9	1787.2	2317.9
(5,5)	3363.7	3752.5	3403.7	3792.1
(7,3)	3972.5	3295.2	3891.1	3357.3
(9,1)	4393.7	1971.5	4127.7	1985.3

Table 1: Percentage improvement of Fair locking (spinning and non-spinning) over the base *sysv* throughput

while all 8 user level locking cases fell within 84.6% and 87.9%. When the $(nlht + lht)$ was increased to $10\mu secs$, the efficiency of *sysv* was still only 82.2%, while those of the user level locks ranged from 98.9% to 99.1%.

When executing this setup with two tasks and two locks the efficiency of *sysv* drops to 18.3% from 25.1% indicating a hot lock in the kernel. At the same time the user level primitives all remain in the same range, as expected. The same effect can be described as follows. With this setup we would expect twice the throughput performance as compared to the 1 task, 1 lock setup. Indeed, for all user primitives the scalability observed is between 1.99 and 2.02, while *sysv* only shows a scalability of 1.51.

In the next set of experiments we fixed the total loop execution time $nlht + lht$ to $10\mu secs$, however we changed the individual components. Let $(nlht, lht)$ denote a configuration. Four configuration are observed: (0,10), (5,5), (7,3), (9,1). The (0,10) represents the highly contended case, while (9,1) represents a significantly less contended case. The exact contention is determined by the number of tasks accessing a shared lock. Contention numbers reported are all measured against the fair locking version of ulocks in a separate run. The contention measurement does not introduce any significant overhead.

Figures 1..5 show the comparision of the 9 locking primitives for the four configurations under various task counts (2,3,4,100,1000). The percentage improvements for each configuration and task count over the *sysv* base number for that configuration are reported in Table 1 for the fair futexes and ulocks without and with spinning (3 $\mu secs$) and in Table 2 for the regular futexes and ulocks.

The overall qualitative assessment of the results presented in these figures and tables is as follows. First comparing the fair locking mechanisms, fair ulocks, in general, have an advantage over fair futexes. Furthermore, fair futexes perform worse than *sysv* for high contention scenarios. Only in the high task count numbers do fair futexes outperform (substantially) *sysv* and fair ulocks. Spinning only showed some decent improvement in the low contention cases, as expected. For the regular versions (ca-locks), both futexes and ulocks always outperform the *sysv* version. The general tendency is for ulocks to achieve their performance at the `(5,5)` configuration with little additional benefits. Though futexes in general lack the ulock performance at the `(5,5)` configuration, they outperform ulocks at the `(7.3)` and the `(9,1)` configurations. In contrast to futexes, spinning for ulocks does not help.

Figure 1 shows the results for 2 tasks competing for 1 lock under four contention scenarios. The lock contention for the 4 configurations were 100%, 97.8%, 41.7% and 13.1%. The lock contention observed for Figure 2.. 5 are all above 99.8%.

Conf	no-spin		spin	
	futex	ulock	futex	ulock
		2 tasks		
(0,10)	8.8	7.6	9.3	7.8
(5,5)	17.7	127.8	86.0	108.2
(7,3)	33.2	60.1	68.5	55.7
(9,1)	40.8	30.9	44.9	29.3
		3 tasks		
(0,10)	43.2	9.0	38.5	9.3
(5,5)	49.1	116.0	89.9	76.5
(7,3)	35.0	38.0	58.0	28.1
(9,1)	39.5	12.8	43.3	12.3
		4 tasks		
(0,10)	61.2	38.8	59.7	33.7
(5,5)	66.6	130.5	116.3	90.5
(7,3)	34.7	29.9	49.1	20.3
(9,1)	36.1	10.5	39.6	6.2
		100 tasks		
(0,10)	456.8	397.1	426.9	399.7
(5,5)	852.3	1030.2	973.4	844.5
(7,3)	1040.4	1003.9	1175.2	919.5
(9,1)	1223.7	967.7	1260.4	936.5
		1000 tasks		
(0,10)	4591.7	4047.9	3333.1	4055.2
(5,5)	6989.5	9570.0	8583.8	8095.9
(7,3)	9149.7	9427.1	10781.5	8714.6
(9,1)	11569.6	9437.7	11869.9	9223.3

Table 2: Percentage improvement of regular (ca) locking (spinning and non-spinning) over the base *sysv* throughput

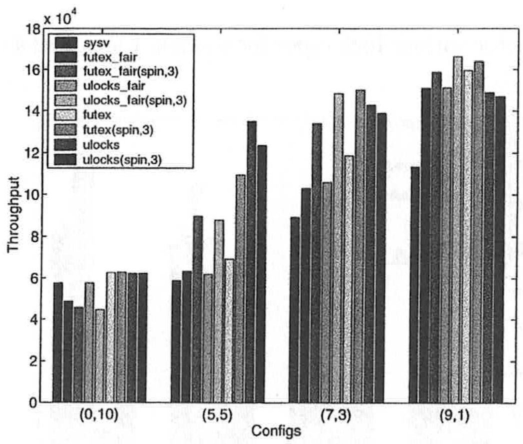


Figure 1: Throughput for various lock types for 2 tasks, 1 lock and 4 configurations

We now turn our attention to the multiple reader/single writer (rwlock) lock primitives. To recall, furwocks implement the rwlock functionality ontop of two regular futexes, while ulocks implement them directly in the interface through atomic compare and exchange manipulation of the lock word. *Ulockflex* allows the specification of a `share-level` for rwlocks. This translates into the probability of a task requesting a read lock instead of a write lock while iterating through the tight loop.

Figure 6 shows the achieved throughput of furwocks and shared ulocks for 2, 3, 4 and 100 tasks competing for a single lock under different read share ratios. The general observation is that the furwocks (solid lines) outperform the ulocks (dashed lines) for their respective task numbers. In general the lower the share level and/or the higher the task numbers the better the improvements that can be achieved with furwocks over shared ulocks. Only in the 100% share-level (only read accesses) do shared ulocks outperform furwocks by $\tilde{2}$-3%.

We now analyze the fairness of the user locking. We monitor the global fairness by computing the coefficient of variance *coeff* of the per task throughput. Note this should not be compared with the fair locking itself. The *coeff* of

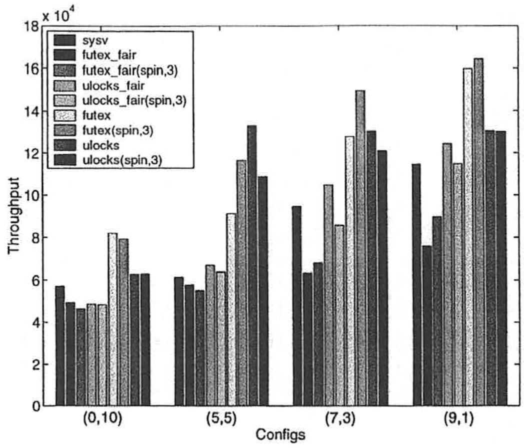


Figure 2: Throughput for various lock types for 3 tasks, 1 lock and 4 configurations

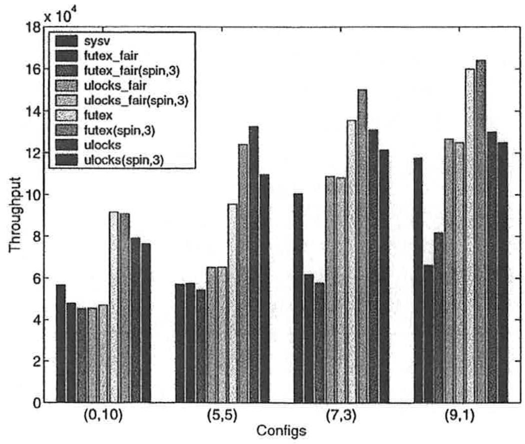


Figure 3: Throughput for various lock types for 4 tasks, 1 lock and 4 configurations

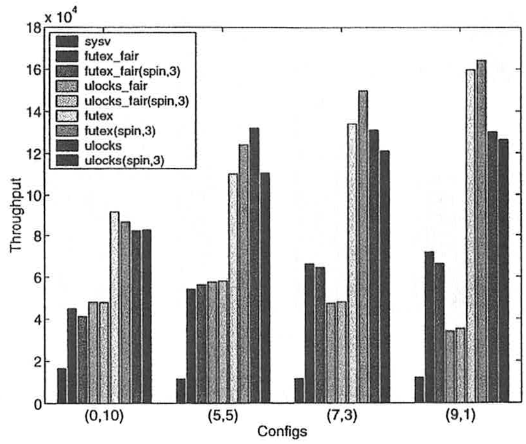


Figure 4: Throughput for various lock types for 100 tasks, 1 lock and 4 configurations

sysv is typically below 0.01. Only the 1000 task case showed a *coeff* of 9.1, indicating that tasks did not all properly get started. The *coeff* for fair futexes and fair ulocks for small task numbers (2,3,4) is in general below 0.01 (as expected). For large task number (100,1000), the *coeff* remains very low for futexes, while ulocks experience a *coeff* as high as 1.10. For furwocks, the general observation is that the *coeff* is less than 0.16 in both furwocks and shared ulocks. Only for the 100 task case does the *coeff* reach 0.45. Overall the mean of coeff for all scenarios is 0.068 for furwocks and 0.054 for shared ulocks. In general we can state that at these level of contention, global starvation is not a problem.

We now turn our attention to the degree of local fairness for the ca-locks. We do this by investigating how many times a task is capable of reacquiring the lock before some other task locks it. To do so, we examine a high contention

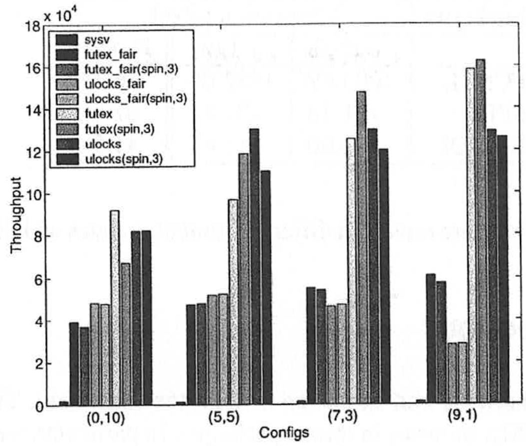


Figure 5: Throughput for various lock types for 1000 tasks, 1 lock and 4 configurations

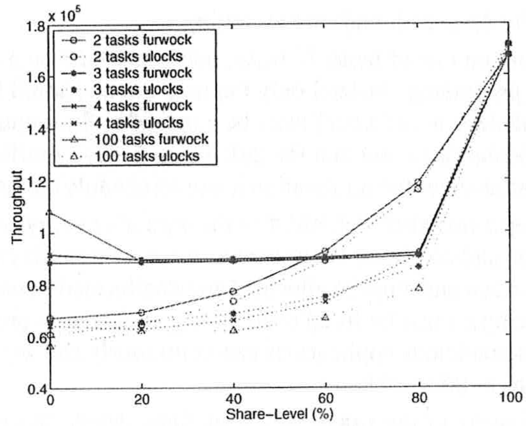


Figure 6: Throughput of furwocks and shared ulocks for (2,3,4,100) tasks competing for a single lock under different read share ratios

case of 100 tasks and the (9,1) configuration. The kernel lock and the fair futexes showed perfect fairness, 99.99% of the task could never reacquire its lock without losing it to some other task. The fair ulocks only 92.1% failed to reacquire, 3.6% was able to grab the lock twice in a row and 0.4% three times. The maximum times a lock was able to be reacquired was 1034 times. For futexes these numbers are 79.0%, 21.0% and maximum of 575 and for ulocks they are 82.4%, 17.54% and maximum of 751. To some degree it confirms that futexes and ulocks have a higher degree of instant reacquisition, however this analysis fails to shed more light on why futexes are so much better than ulocks.

TDB Torture Results

The Trivial DataBase (TDB) is a simple hash-chain-based on-disk database used by SAMBA and other projects to store persistent internal data. It has a similar interface to the classic dbm library, but allows multiple readers and writers and is less than 2000 lines long. TDB normally uses fcntl locks: we replaced these with futex locks in a special part of the memory-mapped file. We also examined an implementation using "spin then yield" locks, which try to get the lock 1000 times before calling yield() to let other processes schedule.

tdbtorture is one of the standard test programs which comes with TDB: we simplified it to eliminate the cleanup traversal which it normally performs, resulting in a benchmark which forks 6 processes, each of which does 200000 random search/add/delete/traverse operations.

To examine behavior under high contention, we created a database with only one hash chain, giving only two locks (there is one lock for the free records chain). For the low contention case, we used 4096 chains (there is still some contention on the allocation lock). For the no contention case, we used a single process, rather than 6. The results shown in Table 3 were obtained on a 2-processor 350MHz Pentium II.

It is interesting that the fcntl locks have different scaling properties than futexes: they actually do much worse under the low contention case, possibly because the number of locks the kernel has to keep track of increases.

Another point to make here is the simplicity of the transformation from fcntl locks to futexes within TDB: the modification took no longer than five minutes to someone familiar with the code.

Locktype	Contention Level		
	High	Low	None
FCNTL	1003.69	1482.08	76.4
SPIN	751.18	431.42	67.6
FUTEX	593.00	111.45	41.5

Table 3: Completion times (secs) of tdbtorture runs with different contention rates and different lock implementations

Current and Future Directions

Currently we are evaluating an asynchronous wait extension to the futex subsystem. The requirement for this arises for the necessity to support global POSIX mutexes in thread packages. In particular, we are working with the NGPT (next generation pthreads) team to derive specific requirements for building global POSIX mutexes over futexes. Doing so provides the benefit that in the uncontended case, no kernel interactions are required. However, NGPT supports a $M : N$ threading model, i.e., M user level threads are executed over N tasks. Conceptually, the N tasks provide virtual processors on which the M user threads are executing.

When a user level thread, executing on one of these N tasks, needs to block on a futex, it should not block the task, as this task provides the virtual processing. Instead only the user thread should be descheduled by the thread manager of the NGPT system. Nevertheless, a `waitobj` must be attached to the waitqueue in the kernel, indicating that a user thread is waiting on a particular futex and that the task group needs a notification wrt to the continuation on that futex. Once the thread manager receives the notification it can reschedule the previously blocked user thread.

For this we provide an additional operator `AFUTEX_WAIT` to the `sys_futex` system call. Its task is to append a *waitobj* to the futex's kernel waitqueue and continue. Compared to the synchronous calls previously described, this `waitobj` can not be allocated on the stack and must be allocated and deallocated dynamically. Dynamic allocations have the disadvantage that the `waitobjs` must be freed even during an irregular program exit. It further poses a denial of service attack threat in that a malicious applications can continously call `sys_futex(AFUTEX_WAIT)`. We are contemplating various solutions to this problem.

The general solutions seem to convert to the usage of a */dev/futex* device to control resource consumption. The first solution is to allocate a file descriptor *fd* from the */dev/futex* "device" for each outstanding asynchronous `waitobj`. Conveniently these descriptors should be "pooled" to avoid the constant opening and closing of the device. The private data of the file would simply be the `waitobj`. Upon completion a SIGIO is sent to the application. The advantage of this approach is that the denial of service attack is naturally limited to the file limits imposed on a process. Furthermore, on program death, all `waitobjs` still enqueued can be easily dequeued. The disadvantage is that this approach can significantly pollute the "fd' space. Another solution proposed has been to open only one *fd*, but allow multiple `waitobj` allocations for this *fd*. This approach removes the fd space pollution issue but requires an additional tuning parameter for how many outstanding `waitobjs` should be allowed per fd. It also requires proper resource management of the `waitobjs` in the kernel. At this point no definite decisions has been reached on which direction to proceed.

The question of priorities in futexes has been raised: the current implementation is strictly FIFO order. The use of nice level is almost certainly too restrictive, so some other priority method would be required. Expanding the system call to add a priority argument is possible, if there were demonstrated application advantage.

Conclusion

In this paper we described a fast userlevel locking mechanism, called *futexes*, that were integrated into the Linux 2.5 development kernel. We outlined the various requirements for such a package, described previous various solutions and the current futex package. In the performance section we showed, that futexes can provide significant performance advantages over standard System V IPC semaphores in all cases studies.

Acknowledgements

Ulrich Drepper (for feedback about current POSIX threads and glibc requirements), Paul Mackerras (for furwocks and many ideas on alternate implementations), Peter Waechtler and Bill Abt for their feedback on asynchronous notifications.

Bibliography

[1] Philip Bohannon and et. al. Recoverable User-Level Mutual Exclusion. In *Proc. 7th IEEE Symposium on Parallel and Distributed Systems*, October 1995.

[2] Robert Dimpsey, Rajiv Arora, and Kean Kuiper. Java Server Performance: Acase study of building efficient, scalable JVMs. *IBM Systems Journal*, 39(1):151–174, 2000.

[3] Hubertus Franke. Ulocks: Fast Userlevel Locking. Available at http://lse.sourceforge.net.

[4] John M. Mellor-Crummey and Michael L. Scott. Scalable Reader-Writer Synchronization for Shared Memory Multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, February 1991.

[5] NGPT: Next Generation Pthreads. Available at http://oss.software.ibm.com/pthreads.

[6] Michael Scott and William N. Scherer III. Scalable Queue-Based Spin Locks with Timeouts. In *Proc. 11th ACMSIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'01*, 2001.

[7] Robert W. Wisniewski, Leonidas I. Kontothanassis, and Michael Scott. High Performance Synchronization Algorithms for Multiprogrammed Multiprocessors. In *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95*, 1995.

[8] Message-ID: <Pine.LNX.4.33.0201071902070.5064-101000@sphinx.mythic-beasts.com>.

[9] Message-ID:
<20020211143841.A1674@elinux01.watson.ibm.com>.

[10] Message-ID:
<E16gRe3–0006ak–00@wagner.rustcorp.com.au>.

[11] Message-ID:
<20020106183417.L10326@holomorphy.com>.

Terabytes on a Diet

Peter Chubb

Introduction

You can buy a multiTerabyte raid array off the shelf nowadays. But it's not much use if you can't plug it into your trusty Linux box.

Although the block layer is in flux, there's still a lot of careless coding that means:

- Even 64 bit platforms are limited to 1 or 2 Tb filesystems (use of 32-bit signed type to hold sector number; sector size hard-coded to 512 bytes)
- Even where the partitioning scheme allows partitioning of larger discs (e.g., EFI's GPT), other limitations prevent them from being used to their full capacity
- Even though the page-cache limit is 16Tb with 4k pages (and indeed if you can create a file this big you can read and write it!) you can't have a filesystem that big.

So...

I set out to remove these limitations on both 64 and 32 bit platforms. But how do you test support for huge (>2TB) filesystems under Linux when the biggest disc you have is 100G? Simple, write a simulator, and use a sparse file for the disc contents. But... it's not that simple.

The problem

Discs are getting bigger and bigger. Figure 1 (from the SCSI industry trade association, http://www.scsita.org/statech/01s005r1.pdf) shows ever bigger discs in the short-term future. Even though this graph was created in 1992, the year 2000 disc sizes and speeds are pretty close to spot-on. If Moore's law continues to hold, we'll have Terabyte[1] discs in our high-end desk-top machines within 5 years.

The only problem is that Linux at present doesn't support large discs. The limitations lie in several places (see figure 2).

- The size in kilobytes of a block device is held in an `int`, which means the maximum size that can be held is $2^{31} \times 1024$ bytes — or 2TB.
- The size of a partition is kept as `unsigned long`, which means that on 64-bit platforms, a partition can be larger than the device that can hold it!
- The SCSI and ATA drivers use the ten-byte command set, which means that the maximum sized disc that can be accessed is $2^{32} - 1 \times$ hardware_blocksize. Many discs use 512 or 1024 byte hard sectors, so the limit is 2TB or 4TB for those discs.
- The standard ext2 filesystem layout restricts the maximum size of a block device to circa 2TB if 1k blocks are used, or circa 16TB (16384GB) if 4kB blocks are used.

[1] Throughout this document, Terabytes, Exabytes etc., are *binary* Terabytes.
The correspondence is as follows:

Prefix		value
Mega-	2^{20}	1 048 576
Giga-	2^{30}	1 073 741 824
Tera-	2^{40}	1 099 511 627 776
Peta-	2^{50}	1 125 899 906 842 624
Exa-	2^{60}	1 152 921 504 606 846 976

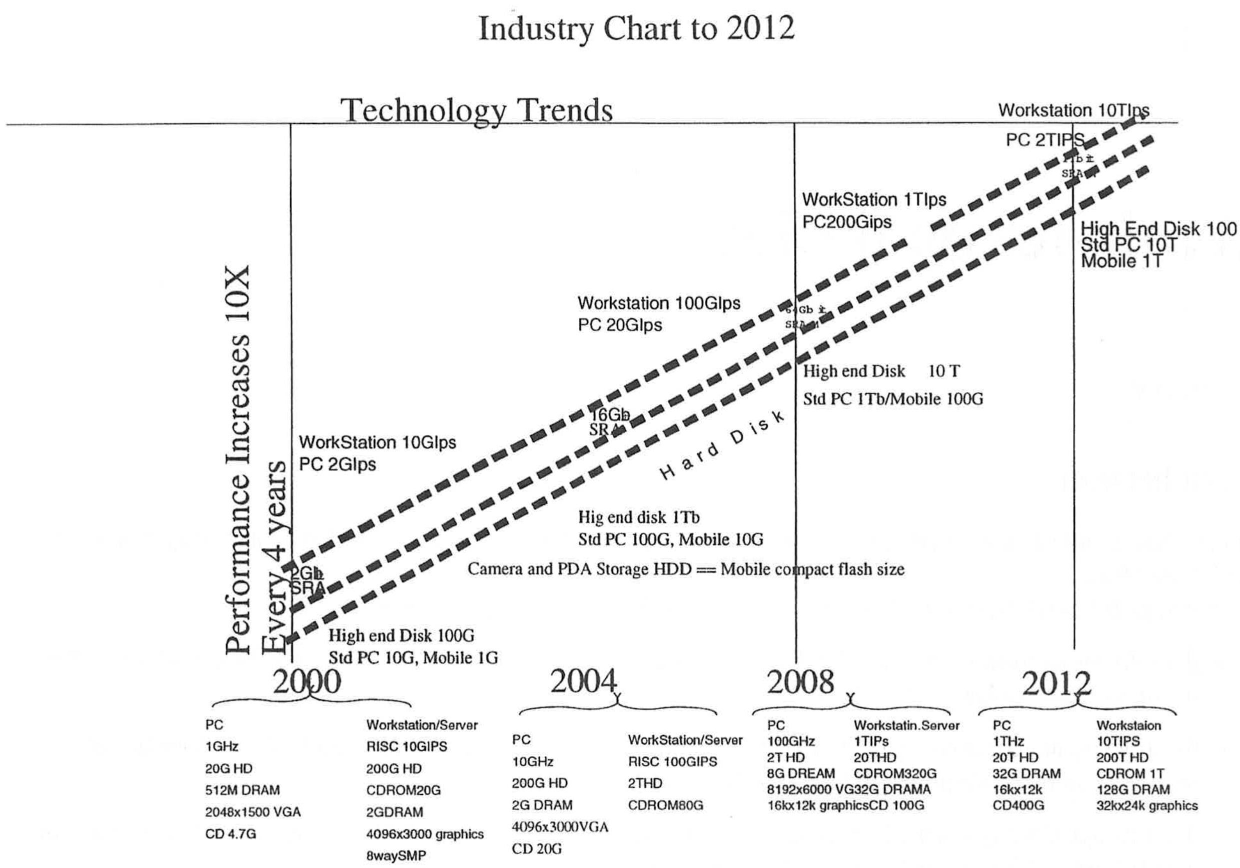


Figure 1: How discs are getting bigger

- All file system operations go through the page cache, which maps a file and an index to a chunk of memory. The index is an unsigned long, and the chunk is currently one page (4k on many platforms), which with 32-bit longs means the absolute largest addressable offset within a file or block device is 16TB.
- Because of these limitations, the user-mode utilities (`mkfs`, `fsck`, etc) have never been tested with large files, and so there's no guarantee that they'll work — and in fact they do not.

Some Solutions

In February 2002, Jens Axboe, one of the block-layer maintainers, introduced a new type `sector_t` into the 2.5 kernel. The intent was that this type should be used wherever sectors or blocks were counted or indexed. On 64-bit platforms, it was 64-bits; on 32-bit platforms, 32 bits now, and possibly 64 bits later on. However, it has not been used consistently.

The places sizes of discs and partitions are stored are:

1. In the `struct gendisk` there's an array of `int`, indexed by device minor number. This array holds the device size in kilobytes.
2. Also in `struct gendisk` there's an array of `struct hd_struct` indexed by minor number; `struct hd_struct` contains the size and offset of each partition in sectors, stored as `unsigned long`
3. Some of the drivers (e.g., the ataraid driver, the 'old' hd driver, etc.) declare a static array of `int` that is eventually pointed to by a `struct gendisk`. Others allocate the array with `kmalloc()`.
4. There is an array `int *blk_size[]` indexed by major and minor device number that contains the size of each disc. this is, as far as I'm aware, used directly only by the `blkdev_size_in_bytes()` function.

All these had to be changed to use `sector_t` instead of `int`, as did the access functions that are used to extract the starting sector and size for each partition.

The structure used to request a block from the block layer `struct request` already used a `sector_t` at the time I started work.

The partition recognition code was also changed to return partition offsets and sizes in `sector_t`. The only partitioning scheme that that currently helps is the EFI GPT scheme, which uses 64-bit integers on-disc to mark out the partitions.

Other partitioning schemes use `unsigned long` if you're lucky, `int` if you're not, and in any case use 32-bit on-disc numbers.

The final changes were to the SCSI get-capacity command. Code like

```
sdkp->capacity = 1 + ((buffer[0] << 24) |
                      (buffer[1] << 16) |
                      (buffer[2] << 8) |
                      buffer[3]);
```

where `buffer` is an array of `unsigned char` had to be changed to cast `buffer[0]` to `unsigned` explicitly, so that the compiler didn't convert it to an `int` then sign extend.

Preliminary Testing

The first thing I did was to redefine `sector_t` to be a struct, and change the obvious places where disc sizes were stored from `int` to `sector_t`, so that the compiler would show me all the places it was used, so they could all be found and fixed. Then `sector_t` could be made back into an integral type.

Having made the changes, the first thing was to see that the result worked on a 32-bit system without enabling the large block device code (i.e., with `sector_t` a `unsigned long`).

After fixing a few typos, the result worked!

OK, create a large (15Tb) sparse file, then mount it via the loop device. I found at this point, that even though writes to the device appeared to succeed, all reads failed. It turns out that if you want a block device to work, its size in sectors must fit into a `sector_t`. So I fixed error handling on the loop device.

It was now possible to create up to a 2TB (sparse) file, and create a file system on it and mount it via the loop device. Attempts to use the loop device on larger files now returned sensible errors. (I still had access to only 100G of disc).

Then the next step was to enable `CONFIG_LBD` on i386 (which turned a `sector_t` into a 64-bit unsigned type), rebuild and reboot, then see what happened.

Now, writes succeed to the loop device using a sparse file up to 16 TB minus one byte, and one can read back what one has written (and it's the same!) Hurrah.

Next was to test `mkfs` for different file systems. All the filesystems tested had essentially the same bug: they did not obtain the actual size of the device. This is because they all used virtually the same code:

> Call `ioctl(fd, BLKGETSIZE, &sz);`
>   If it fails,
>   call `ioctl(fd, FDGETPRM, &x)`
>   if it fails,
>   do binary search to find end of partition

and if these all failed, (because, for instance, the size was held in a 32-bit integer that wasn't big enough) didn't notice, and tried then to create a filesystem on a device of negative size, or that was very small (depending on whether the 32-bit integer was signed or unsigned).

There were two problems here:

1. For a start, `BLKGETSIZE` returned an overflowed 32-bit number if the number-of-blocks wouldn't fit. After fixing this in the kernel,

2. the binary-search overflowed its offset

I fixed the latter by using the `BLKGETSIZE64` ioctl, which is supposed to return the size in bytes of a block device. As it turns out, it'd probably be better to just seek to the end of the device, and return the resulting offset (perhaps trying to read from the last block to make sure it's really there), as `BLKGETSIZE64` has different ioctl numbers different versions of Linux. I'm not sure why the binary search is used in all these `mkfs` (and `mkswap`, for that matter).

Ext[23]

After fixing `mkfs`, I created a large sparse file (16TB), and ran `mkfs` on it. After half an hour or so, it was still trying to write out the same inode group.... and the console was going crazy.

`mkfs` doesn't expect to fail writes because the disc has filled up. (On a *real* disc, it can't happen) (remember I had only 100G of disc).

The ext2/ext3 filesystem layout uses quite a lot of metadata — inodes and bitmaps are written to the disc at `mkfs` time. On a disc of any size, one inode per 4Megabytes of disc. Fortunately this can be controlled by a flag, to allow a sparse file on limited disc storage to be an ext2 file system.

JFS

On IA32, JFS worked like a charm (once the initial problems in mkfs were fixed). The JFS maintainers are quite responsive, and even though I found three bugs (two in mkfs, and one related to page size not equal to block size) — they're all fixed now.

The Loop Device

The loop device is a way of making a file or a block device appear as a different block device (`/dev/loop`*n*). As data is transferred through the loop device it can be massaged in various ways, e.g., to add transparent encryption/decryption. In addition, because a file can be made to appear as a block device, a file containing a file system image can be mounted into the file system hierarchy.

The loop device required some degree of surgery. Unfortunately, there are modules not distributed with the kernel, that do encryption etc., that are going to break with the interface changes necessary to do large block devices.

At present, I've just changed the interfaces, but probably ought to add new ones in parallel with the existing interfaces.

Fake SCSI

One way to check that the SCSI layer is working properly, is to put a simulation of a SCSI host adapter on top of a loop-like device structure. Using real SCSI adapters at present is not an option, because none of them have (yet) been audited for 64-bit cleanness (in fact, the ones I'm using to test 2.4TB file systems will work only up to 2^{32} 512byte sectors.)

There are two SCSI simulators in the Linux kernel. There is one that uses a small (8M) memory region as a disc (`CONFIG_SCSI_DEBUG`) and one that is designed to run on top of the Itanium simulator (`CONFIG_HPSIM` and `CONFIG_SIMSCSI`). I grabbed the latter, moved it into the `drivers/scsi` directory, and hacked at it with a large axe until it used Linux kernel services rather than the simulator services. Because the driver was meant only as a debugging aid, it uses extremely dubious code (I wrote `kernel_write()` to go with `kernel_read()`, modified both to throw away some error checking, and added a custom `loopscsi_open()` routine as well.

This allowed me to check that the partitioning code worked, and that large partitions could be created and recognised by the kernel.

Current Status of the patch

With my patch I see this:

```
SCSI device sda: 18446744073395863552 512-byte hdwr sectors (3783947180439 MB).
```

Without the patch on the same hardware I see this:

```
SCSI device sda: -313688064 512-byte hdwr sectors (-160607 MB)
```

On IA32, (Linux 2.5.24) I've tested ext3, reiserfs and JFS on linear software RAID up to 3.5TB on real hardware, and up to 15TB on the loopback device. All three filesystems work.

On IA64, (Linux 2.5.18 plus IA64 patches), I've tested ext2, reiserfs and JFS on a 2.4TB linear software RAID, but only ext2 works. I didn't test ext3 because of the well-known data-corruption problems in 2.5.18 with ext3.

JFS has issues with page sizes greater than 4k, that I believe are fixed in the current (1.0.20) release. (Even without the LBD patch, JFS from the 2.5.18 kernel would not work on this platform).

Reiserfs wouldn't work either. On mounting, one gets the error:
`reiserfs_fill_super: unable to read bitmap` because the filesystem tries to allocate a chunk of memory 173656 bytes long using kmalloc. Kmalloc thinks this is too large (the largest chunk it'll allocate is 131072 bytes).

Thus Reiserfs is limited at present to 1073709055 blocks (just under 4TB) on a 32-bit system and 536838143 blocks on a 64-bit system (just under 2TB).

The filesystem developers are responsive, and I expect all these problems to be fixed by the time this paper is presented.

You can fetch the patch from `http://www.gelato.unsw.edu.au/patches`

Whether or not the patch makes it into the mainline kernel is in one way irrelevant — what it has done is to raise awareness amongst all the developers of what the issues are, and how to work around them. Recent patches that *have* made it in have used `sector_t` correctly, and don't get in the way of future large-block-device work.

Acknowledgements

The work described here was financed by Hewlett-Packard and the University of New South Wales through the Gelato organisation (`http://www.gelato.org`).

Testing hardware was provided by the Open Systems Development Laboratory (`http://www.osdl.org`).

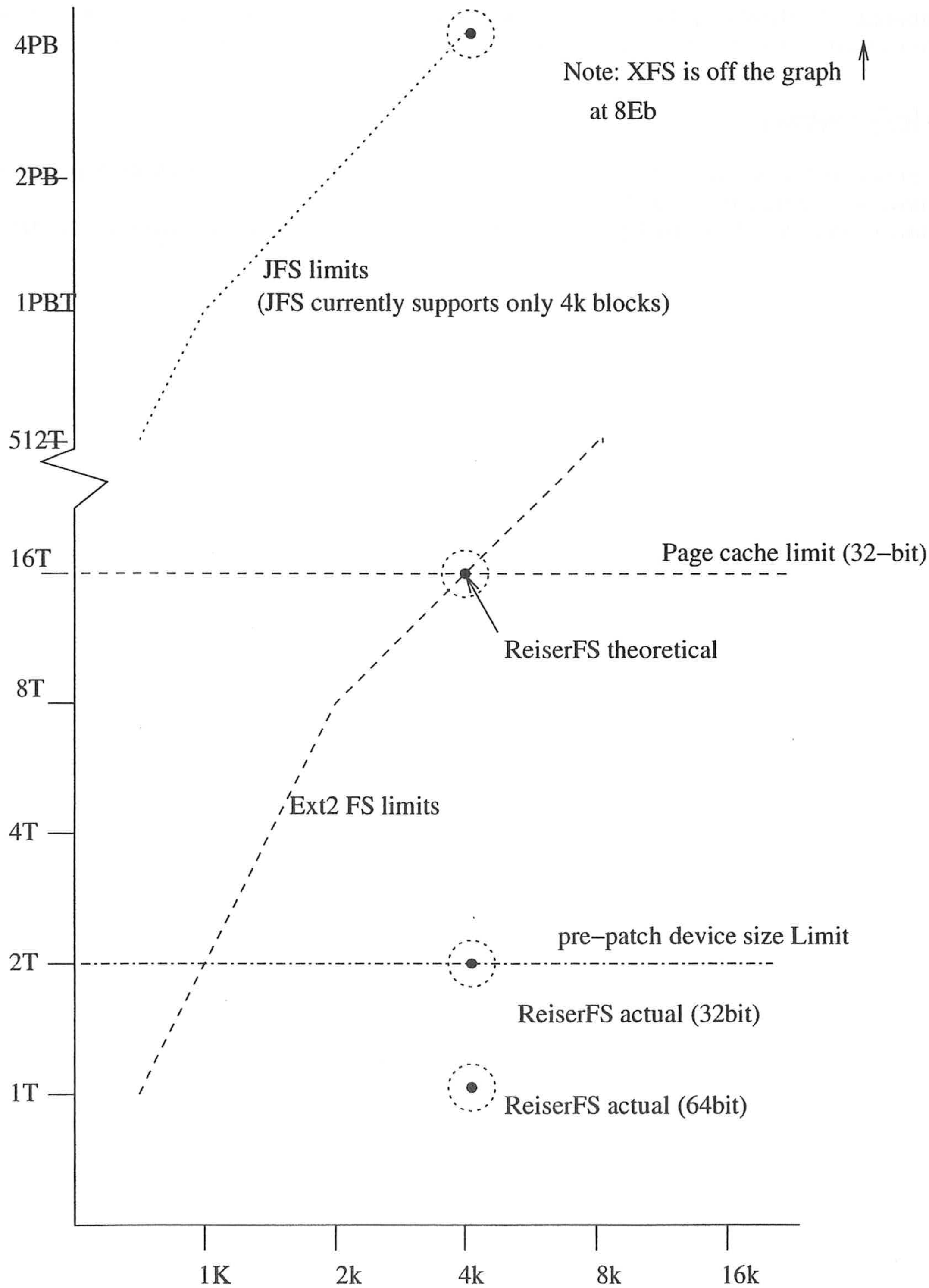


Figure 2: Block device size limitations in Linux 2.4

The Seven Second Kernel Compile

Anton Blanchard
IBM OzLabs Linux Technology Center
<anton@au.ibm.com> · <anton@samba.org>

Abstract

Kernel hackers like to optimise for the common case, and what could be more common to them than compiling kernels.

Timing a kernel compile is often a useful (if somewhat unscientific) metric when making Linux kernel changes. It has been used more recently to identify scalability problems - thus began the kernel compile benchmark wars.

This paper looks at the work required to achieve a seven second kernel compile using the 2.5 Linux kernel on a 32 way PowerPC64 machine as well as the ongoing scalability work to go below seven seconds on this important benchmark.

Introduction

The kernel compile benchmark is a benchmark often used by Linux kernel developers to assess the performance of changes they make. It has the the advantages of being both easy to run as well as having reasonably repeatable results. Although heavily CPU bound, it stresses a number of areas in the kernel, like the process, filesystem and virtual memory subsystems.

Let the wars begin

Martin Bligh kicked the latest round of kernel compile benchmarks off with the following email:

```
From: Martin J. Bligh <fletch@aracnet.com>
To: lse-tech@lists.sourceforge.net
Cc: linux-kernel@vger.kernel.org
Subject: [Lse-tech] 23 second kernel compile (aka which patches
         help scalibility on NUMA)
Date: Fri, 08 Mar 2002 21:47:04 -0800

"time make -j32 bzImage" is now down to 23 seconds.
(16 way NUMA-Q, 700MHz P3's, 4Gb RAM).

...
```

Martin managed to reduce the compile time from 47 seconds on the standard 2.4.18 kernel down to 23 seconds with the help of a number of patches - an impressive achievement. His email served to highlight the current problems with the 2.4 kernel on large NUMA machines and the gains that could be made by adding NUMA memory allocation and scheduling policies into the kernel.

Benchmark Hardware

The PowerPC benchmarking was undertaken on an IBM pSeries p690 server. These are POWER4 based servers which implement the PowerPC AS architecture. From a userspace point of view they are binary compatible with the 32 bit PowerPC architecture and existing PowerPC Linux distributions can run on these machines with only a change of kernel.

POWER4 architecture

A POWER4 chip contains two processors which share a level 2 cache, running at frequencies of up to 1.3GHz. Four of these chips can be combined into a multi chip module (MCM) forming an eight way SMP. There is an each way interconnect between these four chips and the level 3 cache; memory and IO sits behind the MCM.

A p690 combines up to 4 MCMs to form a 32 way SMP and a bus connects these MCMs together. [1]

Logical Partitioning

Logical partitioning allows a single machine to run a number of operating systems. For example a p690 can run up to 16 different operating system instances, either AIX or Linux. [2]

Logical partitioning introduces a layer above the operating system called a hypervisor. This provides a virtualisation layer between the hardware and operating system, allowing it to share resources. It also provides protection between operating systems such that one operating system cannot harm any of the others.

The p690 can run with logical partitioning on or off and all but the final run was done with it enabled.

The PowerPC64 empire strikes back

An initial run on a 24 way p690 resulted in the following post:

```
From: Anton Blanchard <anton@samba.org>
To: lse-tech@lists.sourceforge.net
Cc: linux-kernel@vger.kernel.org
Subject: [Lse-tech] 10.31 second kernel compile
Date: Wed, 13 Mar 2002 19:52:17 +1100

Let the kernel compile benchmarks continue!

hardware: 24 way logical partition, 1.1GHz POWER4, 60G RAM
kernel: 2.5.6 + ppc64 pagetable rework
kernel compiled: 2.4.18 x86 with Martin's config
compiler: gcc 2.95.3 x86 cross compiler

# MAKE="make -j14" /usr/bin/time make -j14 bzImage
...
130.63user 71.31system 0:10.31elapsed 1957%CPU
        (0avgtext+0avgdata 0maxresident)k
```

Thus the yardstick was 10.3 seconds. A number of interesting results could be distilled from the above information:

- Averaged over the entire run, less than 20 CPUs (19.57) were used
- The benchmark was significantly system bound (130.64 user vs 71.31 system)

While most of the benchmark could be split across all CPUs, the final link stage was single threaded. This explained why we saw only 20 CPUs utilised. To answer the second question however, a closer look at the kernel was required.

A look inside the kernel

A simple but often very effective way of obtaining a breakdown of system time on a Linux system is to use the builtin profiler. To enable it a command line option is passed to the kernel: `profile=2` where 2 is the shift count applied to the address of each sample. As all instructions in the PowerPC64 architecture are 4 bytes, a shift count of two will result in a profile with single instruction resolution.

A userspace utility: `readprofile` is used to read the profiling information. The results showed a severe problem in `__hash_page` and to a lesser degree `local_flush_tlb_range` and `local_flush_tlb_page`:

```
201150 total
129051 idled
 43586 __hash_page
  6714 local_flush_tlb_range
  2773 local_flush_tlb_page
```

```
2203 do_anonymous_page
2059 lru_cache_add
1379 __copy_tofrom_user
1220 hpte_create_valid_pSeriesLP
1039 save_remaining_regs
 871 do_page_fault
 575 plpar_hcall
```

In the above readprofile output, column one shows the number of timer hits and column two shows the function that was being executed when the timer tick fired. Each CPU has its own local timer, so these results give a summary of where all CPUs spent their time.

An extension to readprofile by Andrew Tridgell takes the per instruction profiling information and places it on top of a disassembly of the function. In this way timer ticks can be attributed to particular instruction sequences and then back to the source code in question.

As a result of this analysis, the problem with `__hash_page` was clear. It was caused by contention on the `hash_table_lock` spinlock. In order to understand what the `hash_table_lock` protects, a quick background in Linux memory management is required.

2.5 Development

Linux memory management

The 2.5 Linux can run on 13 architectures and so must have a clean abstraction that every memory management unit can be hidden behind. Linux does this by always using a tree representation for pagetables even when the hardware does not. On some architectures like x86, the tree representation is shared by the hardware, whereas on other architectures it is used by Linux only.

PowerPC64 has a hashtable based MMU which is not compatible with the Linux pagetable approach. As a result the PowerPC64 hashtable is treated as an extended TLB with Linux software pagetables backing it. Whenever a Linux pagetable entry is modified, architecture specific code is called to keep the PowerPC64 hashtable in sync. The architecture specific code must provide its own locking since the generic code does not.

PowerPC64 memory management rework

In the 2.4 kernel, all operations on the PowerPC64 hashtable are serialised with a global lock. This lock is the `hash_table_lock` and as seen above is a significant bottleneck on large SMP.

While the global spinlock approach is easy to verify it was obvious that a different approach was required to improve SMP scalability in 2.5. The approach taken was to use a spare bit in each hashtable entry as a per PTE lock. Each operation on a hashtable entry first atomically sets this bit and if it succeeds it can continue to modify the entry. If it fails to atomically set the bit it must back off because another CPU is currently modifying the entry.

TLB invalidation batching

The PowerPC64 architecture requires a sequence of instructions to invalidate a TLB entry:

```
ptesync
tlbie
eieio
tlbsync
ptesync
```

The ptesync, eieio, tlbsync and ptesync are all synchronising instructions. The actual TLB invalidation is done with the tlbie instruction. This five instruction sequence can take a significant amount of time and it is possible to batch invalidations up:

```
ptesync
tlbie 1
tlbie 2
...
eieio
tlbsync
ptesync
```

In doing so we incur the overhead of the synchronising instructions once. Support in the generic Linux kernel is required to batch TLB invalidations and the 2.5 kernel has something called a "TLB gather" which does just this.

The PowerPC64 architecture also requires there to be only one outstanding TLB invalidate on the system at any time. As such all invalidations have to be serialised by a global spinlock.

There are two potential problems with such a global spinlock. Firstly if many CPUs are trying to get the spinlock, they will waste CPU cycles waiting for the lock to become free. Clearly nothing can be done about this problem since it is required by the architecture.

The second problem with a global spinlock is the potential for it to be bounced from CPU to CPU. It can take a large amount of time for a lock to move from one CPU to another and so a lock with an otherwise low amount of contention can become a bottleneck.

The solution for the second problem is to reduce the number of spinlock acquisitions. The TLB batching interface above was modified so the spinlock was taken once at the start of the invalidations and dropped at the end. This addresses the problem with `local_flush_tlb_range` and `local_flush_tlb_page`.

This set of changes plus the addition of eight more CPUs gave us a respectable 7.52s compile:

```
From: Anton Blanchard <anton@samba.org>
To: lse-tech@lists.sourceforge.net
Cc: linux-kernel@vger.kernel.org
Subject: [Lse-tech] 7.52 second kernel compile
Date: Sat, 16 Mar 2002 17:15:35 +1100

> Let the kernel compile benchmarks continue!

I think Im addicted. I need help!

In this update we added 8 CPUs and rewrote the ppc64 pagetable management
code to do lockless inserts and removals (there is still locking at
the pte level to avoid races).

hardware: 32 way logical partition, 1.1GHz POWER4, 60G RAM
kernel: 2.5.7-pre1 + ppc64 pagetable rework
kernel compiled: 2.4.18 x86 with Martin's config
compiler: gcc 2.95.3 x86 cross compiler

make[1]: Leaving directory '/home/anton/intel_kernel/linux/arch/i386/boot'
128.89user 40.23system 0:07.52elapsed 2246%CPU (0avgtext+0avgdata
0maxresident)k 0inputs+0outputs (437084major+572835minor)pagefaults 0swaps
...
```

Current Record

The latest results were done on a 32way pSeries p690 with 1.3GHz CPUs and logical partitioning turned off (only one operating system was running):

```
105.02user 14.50system 0:04.83elapsed 2474%CPU
        (0avgtext+0avgdata 0maxresident)k0inputs+0outputs
        (394245major+570713minor)pagefaults 0swaps
```

4.8 seconds should keep even the most impatient of kernel hackers happy.

Conclusions

The kernel compile benchmark has been a useful benchmark for isolating PowerPC64 Linux scalability issues. With the latest results the challenge is now open for other architectures to better it. And finally, it's cool to have the worlds fastest Linux kernel compile machine. :)

Bibliography

[1] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le and Balaram Sinharoy
POWER4 System Microarchitecture

IBM Server Group, October, 2001
`http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html`

[2] IBM Corporation
Partitioning for the IBM eserver pSeries 690 System
2001
`http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar.html`

Virus Protection for Unix and Open Source Environments

Trend Microsystems
Level 1, 1 Epping Road
North Ryde, NSW, 2113
T 1800 642 421 / + 61 2 9870 4888
F + 61 2 9887 2511
www.trendmicro.com.au

Gateway Scanning

A New Approach to Information Management

The Internet gateway is typically the first point of entry for today's fast-spreading viruses. An antivirus strategy at the gateway is an essential first line of defense. The same technology, which enables seamless communication across the globe, has increasingly become a means of entry for computer hackers.

Virus protection technologies must incorporate management tools that are server-centric and browser-based in order to meet new threats head-on quickly. The mixed environments of most companies today demand a proactive response. A complete antivirus solution should monitor for virus activity and malicious code in SMTP, HTTP and FTP traffic. Scanning for viruses should include all entry points starting with the Internet gateway and ending with individual desktops.

Antivirus is only valuable to an enterprise if it can perform scanning and cleaning functions with minimal impact on resources. It should provide transparent, real-time protection that goes unnoticed by end users. Email messages with attachments should only be scanned if they have the potential for containing viruses. A message should only be scanned once no matter how many users it is sent to within the system. Files and any attachment imbedded within the messages should also be scanned.

A complete antivirus strategy should empower administrators by providing the tools and resources to enforce company antivirus policies. Administrators should have the freedom to customize scanning features to reflect the enterprise's needs, freeing up resources for more business critical activities. It should offer real-time and scheduled full scans with option to launch scans on-demand to address security concerns.

Non-technical end users should not be faced with the task of configuring and updating desktop software. Additionally, an end user should not be able to disable their antivirus protection. Centralized management should put administrators in control of all the network components. Automated tasks should include the updates for virus pattern files, scan engines, or the entire solution from a mouse click.

Trend Micro's melding of intuition and technology has revolutionized the way an enterprise approach virus protection by proving a unique thoroughly integrated antivirus strategy. Trend Micro InterScan VirusWall offers the scalability and stability to fit the needs of an expanding enterprise.

InterScan VirusWall

While a firewall primarily controls outside-to-inside traffic, InterScan supplements this traffic control with anti-virus protection. In addition, InterScan controls *inside-to-outside* traffic. To do this, the suite is implemented in the form of an extranet proxy, which enables control of what users send out (e.g., e-mails with potentially confidential information).

A firewall is node- or machine-oriented and implements security based on a machine's IP address, regardless of who is using the machine. The InterScan extranet proxy is user-oriented–it knows who is on the machine. This level of control means that InterScan can store access properties and rules for different employee groups at different times of the day.

A firewall uses protocol-based security—it bases security decisions on the service being used (i.e., FTP, HTTP, SMTP, or telnet). InterScan supplements this with the ability to implement application-based security, which is based on the *content* of the service, enabling centralized control of the content that users receive. This content can be determined generally via an established company policy, and specifically, by configuration of InterScan.

InterScan was designed to function in high traffic environments. The product's COM/DCOM (Common Object Modeling/Distributed COM) architecture is highly scalable. Each of the *e*Manager functions is an independent object. Due to this, *e*Manager can run on one or any number of machines. DCOM will automatically balance the load. If higher performance is required, the administrator can simply add another PC.

There are two key components of InterScan VirusWall - Email VirusWall and Web VirusWall.

E-Mail VirusWall

The E-Mail VirusWall module is implemented inside a firewall and on the firewall side of your existing SMTP server. The idea is to have E-mail VirusWall listen on port 25 for new connections, scan the SMTP traffic it receives, and then route scanned traffic to your original SMTP server for delivery as usual to the mail clients.

The hallmark of Trend Micro's E-Mail VirusWall module is its patent pending MacroTrap^{TM} technology. This technology supplements traditional pattern matching techniques with sophisticated rule-based scanning, which is particularly adept at detecting macro viruses. Using MacroTrap, macro commands embedded in word-processed and spreadsheet files are analyzed to determine whether the macro's execution would lead to malicious behavior. As a result, even those macro viruses that have not yet been captured and identified can be detected and cleaned.

E-Mail VirusWall intercepts each SMTP file and diverts it to the scanning engine, where the file is checked to determine whether it is capable of carrying a virus. If so, it is compared to an extensive database of virus signatures most likely to cause infection over the Internet. Large data files such as bitmaps and many multimedia files, which cannot carry viruses, are automatically transferred without scanning. The engine also scans 15 types of compressed files, as well as BINHEX, UUENCODE, and MIME formats.

Content Manager

Spam mail blocking and e-mail filtering are both forms of content management that are handled by the Content Manager. The important difference between these two capabilities is that to block spam mail, the administrator sets rules that apply to the entire mail message, whereas in content filtering, the content of the e-mail message itself is scanned for keywords.

Hence, the spam mail portion of the module applies rules such as "if the mail is from domain 'junk.com' and is sent by 'user,' delete the mail". Other configurable actions include message quarantine or archiving. The spam filter contains a list of known spammers that was developed from several lists in the public domain.

In the e-mail filter portion of the module, the administrator can establish a set of keywords, and if any keyword is contained in an e-mail in transit, a pre-configured action is taken for that e-mail. These actions can include quarantining the mail without mail delivery, deleting the e-mail, or archiving the message in a directory but allowing mail delivery. The administrator can configure customized notification in each case to the sender, receiver, and other e-mail users.

Traffic Manager

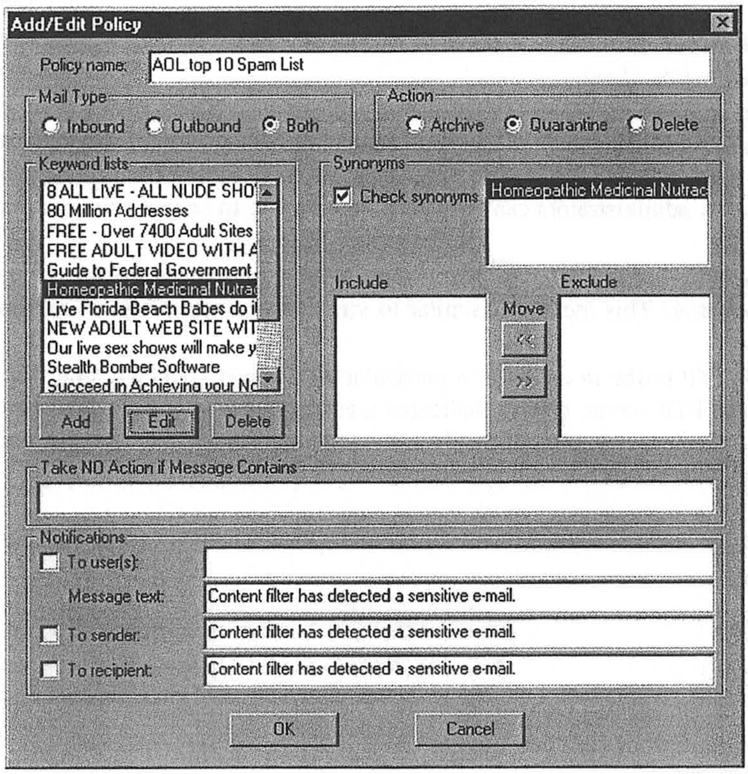
Some administrators manage e-mail by simply imposing an upper limit on e-mail attachment size and returning any e-mail that exceeds this size to the sender. Traffic Manager module provides a more flexible alternative.

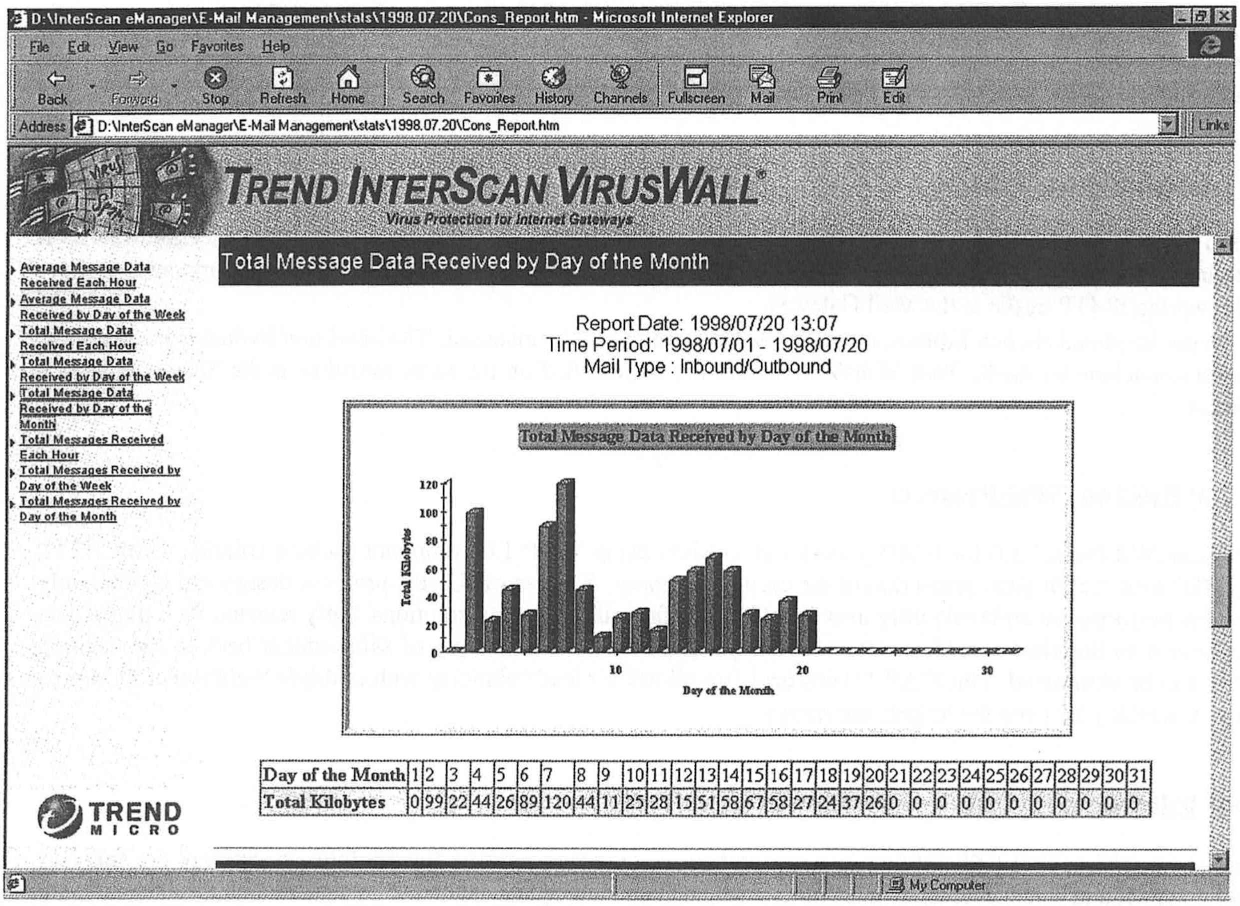
The Traffic Manager contains two parts—an e-mail statistics and usage collector, and a load balancer—that work together to aid the administrator in e-mail management.

The statistics collector provides information about company e-mail usage (e.g., bytes transferred per hour and hourly load times) that is stored in an SQL database and integrated in graphical reports for display or printing. Based on this information, the load balancer allows the administrator to control message delivery times as a function of mail length.

Web VirusWall

The Web VirusWall module detects viruses attached to either HTTP or FTP Web transfers. Virus scanning incorporates MacroTrap technology for HTTP and FTP traffic, as well as malicious code protection for HTTP transfers.


Add/Edit Policy
Policy name: AOL top 10 Spam List
Mail Type
Inbound Outbound Both
Action
Archive Quarantine Delete
Keyword lists
8 ALL LIVE - ALL NUDE SHO
80 Million Addresses
FREE - Over 7400 Adult Sites
FREE ADULT VIDEO WITH A
Guide to Federal Government
Homeopathic Medicinal Nutra
Live Florida Beach Babes do i
NEW ADULT WEB SITE WIT
Our live sex shows will make y
Stealth Bomber Software
Succeed in Achieving your N
Add Edit Delete
Synonyms
Check synonyms
Homeopathic Medicinal Nutrac
Include
Move
Exclude
Take NO Action if Message Contains
Notifications
To user(s):
Message text: Content filter has detected a sensitive e-mail.
To sender: Content filter has detected a sensitive e-mail.
To recipient: Content filter has detected a sensitive e-mail.
OK Cancel


D:\InterScan eManager\E-Mail Management\stats\1998.07.20\Cons_Report.htm - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address D:\InterScan eManager\E-Mail Management\stats\1998.07.20\Cons_Report.htm
Links
TREND INTERSCAN VIRUSWALL
Virus Protection for Internet Gateways
Total Message Data Received by Day of the Month
Average Message Data Received Each Hour
Average Message Data Received by Day of the Week
Total Message Data Received Each Hour
Total Message Data Received by Day of the Week
Total Message Data Received by Day of the Month
Total Messages Received Each Hour
Total Messages Received by Day of the Week
Total Messages Received by Day of the Month
Report Date: 1998/07/20 13:07
Time Period: 1998/07/01 - 1998/07/20
Mail Type : Inbound/Outbound
Total Message Data Received by Day of the Month
Total Kilobytes
Day of the Month
Day of the Month 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Total Kilobytes 0 99 22 44 26 89 120 44 11 25 28 15 51 58 67 58 27 24 37 26 0 0 0 0 0 0 0 0 0 0 0
TREND MICRO
My Computer

Trend Micro recommends you install Web VirusWall inside the firewall and between the Internet and the HTTP proxy. The idea is to have Web VirusWall listen on a port- typically 80- for requests from the HTTP proxy, relay the requests to the remote Web server, and then scan the HTTP traffic it receives in response before passing it on to the proxy and ultimately the requesting client.

In fact, Web VirusWall contains two layers of protection against malicious ActiveX controls and Java applets. First, the scanner recognises Authenticode^{TM} signatures. Matching Java applets ActiveX objects to a known Authenticode signature database deciphers whether these objects come from a trusted source or have not been modified during transmission. Network administrators can configure this module to accept or reject objects based on their source.

Second, the scanner actually scans inside the instruction codes and matches the instructions to a database of known malicious applet patterns. This method is similar to virus pattern matching and relies on frequent pattern updates to remain effective.

If you want to scan all FTP traffic in or out of a particular FTP server (typically one that you host), you can install Web VirusWall to that FTP server, or on a dedicated machine between it and the requesting clients. In this case, it appears to users that they are connecting directly to the target server when in fact they are connecting to Web VirusWall VirusWall, which then relays the request to the specified server.

Available Editions

Standard Edition

In the Standard Edition, each service is a separate daemon. Therefore, all three VirusWalls can be installed on the same machine (e.g., a dedicated server) or each can be installed onto a different machine (e.g., the server for which it will scan, or a dedicated server).

CVP Edition

In the CVP Edition, all three services are included in one daemon. Therefore, all three services will be installed on the same machine. If you want to distribute the tasks among several CPUs, install InterScan on each machine, then control which protocols are scanned using the FireWall-1 rules base.

SendMail Switch Edition

Trend Micro's Sendmail Switch integration allows a third method of deploying InterScan's Internet gateway level antivirus software. Sendmail's Content Filter API offers InterScan VirusWall a standardized programmatic interface for scanning SMTP traffic at the Mail Gateway.

In the Sendmail Switch Edition, only the SMTP daemon will be installed. The Sendmail Switch Edition should run on a machine by itself. Web VirusWall should not be installed on the same machine as the Sendmail Switch Edition.

ICAP Edition (WebProtect)

InterScan WebProtect 1.0 for ICAP provides enterprises using ICAP 1.0-compliant caching solutions with HTTP and FTP-over-HTTP virus protection at the caching gateway. The use of ICAP's protocol design can significantly improve performance and scalability over standalone or firewall-integrated solutions. Only relevant Web traffic data can be sent to the virus scanning server for analysis, and redundant delivery of safe content back to the caching server can be eliminated. The ICAP 1.0 protocol also allows for load balancing with multiple WebProtect servers to allow scalability for even the largest enterprises.

CSP Edition

Trend Micro VirusWall CSP Edition is a comprehensive antivirus solution for the Internet gateway. It analyzes SMTP, HTTP, and FTP traffic as an intermediate step before sending messages and files on to their final destination.

The Content Scanning Protocol (CSP) was defined and developed by Trend Micro to allow partner Internet firewalls to support virus scanning via Trend Micro InterScan VirusWall. Customers using supported firewalls can protect their internal network from virus infections and outbreaks with InterScan VirusWall CSP Edition.

Platform Support

	Solaris	HP-UX	Linux	Tru64	AIX
Mail/SMTP					
Standard+eManager	YES+YES	YES+YES	YES+YES	YES+NO	YES+NO
CVP	YES	NO	NO	NO	NO
SendMail Switch/api	YES	NO	NO	NO	NO
CSP	YES	NO	NO	NO	NO
Web/HTTP					
Standard	YES	YES	YES	YES	YES
CVP	YES	NO	NO	NO	NO
iCap	YES	NO	NO	NO	NO
CSP	YES	NO	NO	NO	NO
File/FTP					
Standard	YES	YES	YES	YES	YES
CVP	YES	NO	NO	NO	NO
CSP	YES	NO	NO	NO	NO

Working InterScan into Open Source SendMail Servers

Integrating InterScan VirusWall with Open Source SendMail can be done, although this process is far more involved and complex technically. A compatible version of Open Source SendMail (8.11 or later) must be downloaded as source code. The source will need to be built with the Content Filter API enabled. Furthermore, the library that provides API callback methods must be built as a shared library. Configuration changes to enable InterScan virus scanning are far more complex to accommodate. This work is only recommended for SendMail experts.

Antivirus software for the Notes environment

How viruses spread within Lotus Notes / Domino

Malicious code is insidious. Without virus detection or protection, users typically will not know that their systems are infected until they see results of a virus payload. This can range from an annoying; irreverent message being displayed on-screen to a catastrophic reformatting of users' hard drives that destroys all their data. Viruses spread through Lotus Notes' four information dissemination components — email, shared databases, replication and Internet/Intranet information sharing.

The following outlines how malicious code spreads in the Notes environment:

- **Email:** Today's office worker receives an average of more than 40 email messages each day. Viruses need to copy and spread themselves, because the text messages of electronic mail are, by themselves, unable to spread viruses. The virus danger from email stems from attachments containing active executable program files with extensions such as: CLASS, OCX, EXE, COM and DLL — and from macro-enabled data files. Microsoft Word and Excel data files have been spreading macro viruses through this capability since 1995 and are today responsible for the majority of all virus infections.

- **Shared databases:** The second mode of Lotus Notes virus transmission involves shared databases. These databases are capable of storing millions of data file archives that are accessible throughout the Lotus Notes / Domino network. The widespread virus problem makes it likely that such broad access to Notes databases will infect users machines which access infected files within these databases.

- **Notes replication:** Replication, which updates changes to local databases to all other databases, is the third way that viruses can propagate throughout a Lotus Notes / Domino network (Figure 1). By synchronising databases at different locations, replication can simultaneously spread to all Domino servers any virus residing in data file in any database.

- **Internet and intranet transfers:** A fourth mode of virus infection and propagation now faces Lotus Notes users. The Lotus Notes' Domino server makes information sharing across Intranets and the Internet easy, opening up entire new worlds of data to the Notes user. For example, browsing the World Wide Web or their company's Intranet, a user can save information found in open databases that anyone can access. This powerful way of storing and later retrieving valuable information can lead to the infection of user's machines. For example, macro viruses can spread through files downloaded from the web via FTP or direct web page access. Since Domino servers are increasingly being used to access the Internet and intranets, web-based virus threats are a larger threat in the Notes environment.

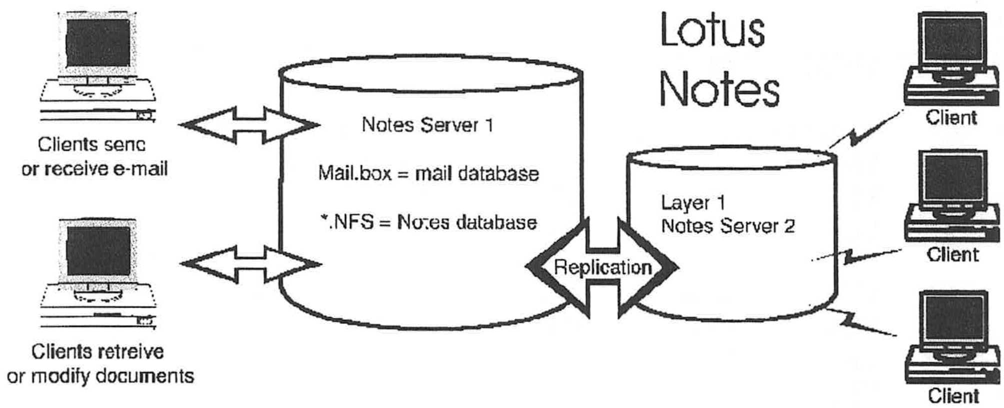


Figure 1: In a Lotus Notes environment, clients can send or receive email, and retrieve or modify documents from a public database. Lotus Notes also synchronizes databases at multiple locations.

Protecting the Lotus Notes/Domino Environment

The corporate-wide approach of monitoring all possible virus infection paths in the network should be extended to the Lotus Notes and Domino email/groupware infrastructure. The challenge is to pinpoint and protect all possible entry points for virus-infected files or other malicious code within each of the network's Lotus Notes databases.

An effective antivirus product for the Lotus Notes/Domino environment must be able to monitor all Notes functions in real-time. It must scan all new entries into the databases, and clean or delete any infected files or inbound malicious code — and do so with a minimal performance impact on the Domino server.

Traditional antivirus products view the Notes database as a single large file, making them unable to locate viruses within the database. Effectively eradicating viruses in the Notes environment requires development of antivirus software specifically tailored to Lotus Notes and Domino. The virus scanner must understand the Notes database format, scan individual files and documents within the database, and clean or remove infected files without impacting the Notes database structure.

Trend Micro was the first company to recognise the need, and with its April 1996 introduction of ScanMail for Lotus cc:Mail, became the first to develop and ship, virus protection for proprietary email environments. In May 1997, Trend introduced ScanMail for Lotus Notes — the first program to operate natively under Notes' proprietary communications environment to provide complete virus protection. ScanMail for Lotus Notes detects infected files, eliminates viruses in real-time as they pass through Notes servers, and scans information archived in Notes servers. Today, ScanMail supports the widest range of operating systems including S390, AS400, NT, Solaris, Linux, AIX and O/S.

How ScanMail Works

In Notes Mail, each piece of mail is first placed in a Notes Mail router, which stores the mail in a virtual mail queue. The Notes Mail scheduler reads the mail from the queue, resolves the address and generates a copy of the mail for each recipient. This scheduler then either deposits a copy of the mail into the recipient's individual mailbox on the same server, or sends it to a router program on another server.

ScanMail for Lotus Notes ties into the mail router, detects new mail as it arrives, instructs the virus scanner to scan the mail queue, and prepares the mail for pickup by the mail scheduler. This approach has several advantages over alternative methods. First, since the mail is received by the router and then scanned, mail sending is not delayed. Alternative approaches retrieve each piece of mail before it reaches the router, introducing a delay prior to routing.

A second advantage also involves the timing of the virus scan. Since this scanning takes place before the scheduler accesses the mail, only one copy of the mail is scanned. This method conserves scanning resources and prevents infected mail from being routed to other servers before scanning. An alternative approach employing an agent to monitor each mailbox often requires scanning of multiple copies of the mail and allows infected mail to be routed to other servers. While Trend Micro's approach is two-way, scanning both in-bound and out-bound message attachments, the alternative approach is solely a one-way scan.

The third benefit of Trend Micro's approach is that the mail routing path remains unchanged. No special mail path or mailbox is used, increasing performance and eliminating false deliveries that are possible in alternative approaches.

ScanMail offers further protection in the form of:

- **Database Protection:** ScanMail for Lotus Notes features two types of protection for data stored in Notes

databases - "On-demand" (manually activated, or pre-scheduled) scans for archived data such as email or old databases, and real-time scanning of documents as they are saved to Notes databases.

On-demand scans, which can also be scheduled to occur automatically, are able to penetrate the special Notes database format and examine all files that are part of the database. Only in this way can the Notes administrator be sure that the database is initially "clean," preventing re-infection each time an infected document is accessed.

Real-time scans are the heart of ScanMail for Lotus Notes database protection. On the Domino server, clients can open a document from the shared database, modify it (possibly infecting it with a virus), and return it to the shared database. Whenever a client saves a document in this database, ScanMail for Lotus Notes scans the document for viruses just before it is closed, preventing the spread of any virus present on the document. And this action is imperceptible to the user. Once the document has been found to be virus-free (or if found to have a virus, once cleaned, quarantined, deleted, or ignored), clients are allowed to access the file.

- **Replication Protection:** In Lotus Domino, when replication begins, the replicator opens the database, updates an entry (i.e., document), closes the entry, updates the next entry, closes that entry, and so on. Whenever a database entry is updated, the real-time replication scan in ScanMail for Lotus Notes scans that document for viruses just before it closes. If a virus is detected in the entry, replication of that entry is blocked. But during this scan, ScanMail allows the task of replicating other entries to continue unabated. Hence, even if a virus is identified during a scan, replication of clean entries continues, and there is no degradation in replication performance. The result is replication of only clean entries. As during the database scan, at the administrator's option, infected files can be deleted, moved, passed, or cleaned.

  Compared to methods that require virus scanning to be completed before replication is continued, this approach saves on costly dial-up telephone connections when replication involves remote servers. In fact, alternative approaches can double or even triple telephone connection time and costs.

- **Domino Server Protection:** To protect Lotus Notes Domino Servers from viruses introduced via FTP downloading from the Internet, Trend Micro's ScanMail for Lotus Notes scans each downloaded file.

File Server Scanning

Although Linux built-in security features may make the operating system more immune to certain types of viruses, Linux-based FTP servers or networked file servers may still be used as a carrier for malicious code. Linux itself is still vulnerable from attacks by worms and Trojans. Worms, malicious code files spreading through direct Internet connections and between networked servers, can exploit known vulnerabilities of the Linux system and open backdoor components that allow hackers access to password and other information through infected Web servers. The "ELF_Ramen.10" worm, detected by Trend Micro in January 2001, was just such a threat. Months later, the ELF_Adore.A worm appeared with the ability to allow a remote user to gain root access and manipulate files.

Since Linux is an open source operating system, there is no control over who obtains a copy of the software kernel. In theory, virus writers and hackers have the ability to thoroughly study this operating system, and may be just as versed in breaking into a system as the system administrator was in creating it.

Trend Micro ServerProtect for Linux 1.0

Recognising the need for robust, dependable virus protection for the open source world and using the knowledge gained from developing and licensing their gateway antivirus technology for Red Hat, SuSE, and Turbo Linux distributions with its InterScan VirusWall product, Trend Micro has created one of the first complete virus protection products for the Linux-based file server. Trend Micro ServerProtect is server-based virus protection that can detect known and unknown viruses, even unknown macro viruses.

ServerProtect is designed to fully protect file servers with minimal performance and administration overhead. Acknowledging the growth and increased complexity of internal networks, ServerProtect also provides antivirus software management functions to reduce administrative costs.

Trend Micro's proven virus scanning technology delivers a powerful tool to protect your electronic assets from virus attacks. ServerProtect for Linux 1.0 supports Red Hat 6.2 and Red Hat 7.1 distributions.

How ServerProtect for Linux Works

ServerProtect for Linux makes use of the following technologies to detect different forms of malicious code:

- **Pattern Matching:** ServerProtect draws upon an extensive database of virus patterns to identify viruses, and other malicious code, through a process called "pattern matching". Key areas of suspect files are examined

for telltale strings of malicious code and compared with thousands of virus signatures that Trend Micro has on record. For polymorphic or mutation viruses, the ServerProtect scan engine permits suspicious files to execute in a protected area for decryption. ServerProtect then scans the entire file, and looks for strings of mutation-virus code.

- **MacroTrap:** Macro viruses are application specific, and are not confined to a particular operating system. So long as an operating supports the macro's application, it can be infected. Given this cross-platform compatibility, combined with the growing popularity of the Internet, and increasing power of macro languages, the magnitude of the threat posed by these viruses is obvious. Trend Micro's MacroTrap is designed to provide you with a means of protecting your network from this threat.

  The MacroTrap performs a rule-based examination of all macrocode that is saved in association with a document. Macro virus code is typically contained as part of an invisible template (e.g. *.dot in Microsoft Word) that travels with the document. Trend Micro's MacroTrap checks the template for signs of a macro virus by seeking out instructions that perform virus-like activity. Examples of this behavior are: copying parts of the template to other templates (replication), and execution of harmful commands (destruction).

- **ScriptTrap:** Script viruses are written in script programming languages, such as VBScript and JavaScript. VBScript (Visual Basic Script) and JavaScript viruses make use of Microsoft's Windows Scripting Host to activate themselves and infect other files. This means that these viruses can be activated simply by double-clicking the *.vbs or *.js file from Windows Explorer – a standard application in the Windows family of operating systems.

  HTML viruses use the scripts embedded in HTML files to do their damage. These embedded scripts automatically execute the moment the HTML page is viewed from a script-enabled browser. Using Script Trap technology, ServerProtect not only guards against harmful known script-based viruses ("I Love You" and "Anna Kournikova"), but can also protect your Linux server from new, unknown script-based threats.

  Using a combination of lexical analysis and semantic parsing, ScriptTrap identifies the actions that the script is meant to perform. Then, based on rules contained in the virus pattern files, tags the script as either malicious or harmless.

- **Compressed File Scanning:** Compressed files and archives (i.e. a single file composed of many, often compressed, files) are the preferred file format for file distribution via email or the Internet. Unless your antivirus application is specially equipped to handle these files, viruses and other malicious code may be "smuggled" into your network inside these files. The scan engine in ServerProtect can scan compressed files and the contents of archives. It can even detect viruses in compressed files and archives composed of other compressed files up to five compression layers.

  The Trend Micro scan engine can detect malicious code in the following compression and archival algorithms:

  - PKZIP (.zip) and PKZIP_SFX (.exe)
  - LHA (.lzh) and LHA_SFX (.exe)
  - ARJ (.arj) and ARJ_SFX (.exe)
  - CABINET (.cab)
  - TAR
  - GNU ZIP (.gz)
  - RAR (.rar)
  - PKLITE (.exe or .com)
  - LZEXE (.exe)
  - DIET (.com)
  - UNIX PACKED (.z)
  - UNIX COMPACKED (.z)
  - UNIX LZW (.Z)
  - UUENCODE
  - BINHEX
  - BASE64

As a system resource conservation measure, ServerProtect does not scan files, within compressed archives, that exceed a specific size for Linux. These files are skipped, and are recorded in the scan logs. Other files in the archive that do not exceed this limit are still scanned.

About Trend Micro

Trend Micro is a global leader in antivirus and content security software and services. Founded in 1988 by Steve Chang, the company led the migration of virus protection from the desktop to the network server and the Internet gateway - gaining a reputation for vision and technological innovation along the way. Today, Trend Micro specializes in high-performance, server-based virus protection for large enterprises, as well as for small to medium-sized enterprises and consumers.

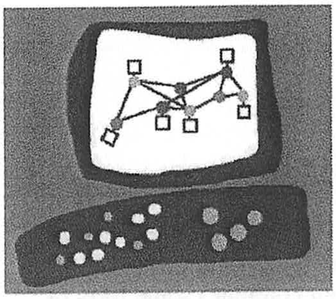
In just over a decade, Trend Micro, headquartered in Tokyo, Japan, has grown into a global organisation, with over 1600 employees in 25 countries. Sales have soared by an average of 75% each year reaching US$241 million in 2001. The company is listed on the Tokyo Stock Exchange and on the NASDAQ. Trend Micro is headquartered in Tokyo, Japan.

Trend Micro has a solid reputation for transforming great ideas into cutting-edge technology. Citing its strategy and vision, the Gartner Group has hailed Trend Micro as the most visionary malicious code management supplier for four consecutive years.

Further Information

- Trend Micro: `http://www.trendmicro.com`
- InterScan VirusWall: `http://www.trendmicro.com/products/isvw/`
- ScanMail for Lotus Notes: `http://www.trendmicro.com/products/smln/`
- ServerProtect: `http://www.trendmicro.com/products/svrprt/`

Security In the Enterprise: Securing Applications in Transition



Jason Loveday
Check Point Software Technologies Ltd

Today, enterprise-wide networking means connectivity to anyone, from anywhere, at any time. It also means making all information available. A typical enterprise might have:

- An e-mail system, providing a mailbox to every employee.
- A payroll system, responsible for delivering the employees' salaries.
- A human resources system, maintaining employee records such as home address, next of kin and other personal information.
- A CRM system with our customer records, etc.

These enterprises IT systems become complex to manage, especially requirements to access multiple applications. Where these applications are often built without security in mind. Security components are usually implemented as an afterthought, by developers whose core competencies are not those associated with information security. Providing an adequate layer of security for these applications in turn becomes prohibitively complex. Users are confronted with many passwords and differing authentication mechanisms, such that simple passwords are often chosen, or even written down on keyboards or monitors. Thus what results is an ineffective security system for our applications.

Data transiting the network infrastructure is community property and anyone connected to the network has access to it. Passwords and other sensitive data travel along this communal infrastructure and can be read by anyone. The integrity of data and other transactions cannot be assured, as it not only can be read, but also manipulated in transit. Passwords sent in clear text across the network can be intercepted and in turn access credentials to business critical applications stolen. Thus the authenticity of transactions cannot be guaranteed. Simply, we do not know if the authorised user is accessing the system since anyone connected to the network has access to these credentials.

Given that data transiting the network is accessible by anyone, authorised or otherwise, it follows that even the best authentication systems can be ineffective. If "Alice" is authorised to use the system, and "Bob" is intercepting the data, the "Bob" effectively has access to the same information as "Alice".

Also, "Alice" may have several credentials for several systems, which carries a significant administration overhead. Administrators are often required to maintain multiple user databases. These disparate databases can be difficult to keep up to date, which introduces additional security considerations.

Securing Applications and Data in Transit

Information Security can briefly be defined as:

- Confidentiality
- Integrity
- Availablility

In addition, an effective enterprise-wide security implementation should meet the following criteria at a minimum...

- Authentication
- Privacy
- Account for users' activities
- Ease user experience

Authentication

The mechanism by which users identify themselves to applications needs to be effective. An effective authentication system is one in which users identities are assured with reasonable certainty. At a minimum, some form of "two factor" authentication system should be implemented. A two factor authentication system combines some form of physical token, "Something I have", with a pass code that only the user knows, "Something I know".

Ideally the user accounts database should be centralized. A single user database is easier to manage than multiple, disparate systems, which in turn provides a more reliable system and improves security.

Privacy

Communications across shared segments of the network should be encrypted, end-to-end, to prevent access by unauthorised users and in turn ensure data integrity. Thus, when "Alice" accesses an application, she is the only user who can view or manipulate the data for that session, and "Bob" cannot gain anything by intercepting the encrypted data.

Account for users' activities

Most security violations occur from within the network, by users with legitimate access credentials. It is very difficult to protect the network against malicious use by authorised users. For example, a disgruntled employee might erase or damage some business critical files, or a defecting sales person might steal the customer database on their way out. In the absence of adequate policies and means to police these there is little that can be done.

In dynamically addressed networks, such as those utilizing Dynamic Host Configuration Protocol (DHCP), the task of tracking user activities is made even more challenging.

Ease User Experience

Users will always find ways around things that annoy them or impact on what they are really doing. It is quite common to see users writing passwords down on sticky notes on keyboards or monitors, or choosing passwords that are easy to guess.

Ideally a user should only have one set of credentials for all systems they access. The security layer implemented across all applications should be as transparent as possible to the end user. Security should not get in the way.

How Hackers Do It: More Tricks, Tools, and Techniques

Alex Noordergraaf
Sun Microsystems
<alex.noordergraaf@sun.com>

Abstract

Understanding the techniques commonly used by hackers is critical to establishing good system security. This paper provides an overview of methods typically used by hackers focused on attacking particular systems or organizations. These methods include the mapping of network topologies (footprinting), identification of host operating systems and available services (fingerprinting), and methods for hiding evidence of intrusion. When unauthorized system access is achieved, a hacker will generally install some type of backdoor on the system to insure that future access is guaranteed; common backdoor mechanisms are also discussed this paper, as an understanding of these mechanisms is necessary when trying to detect previous intrusions and prevent further unauthorized access to a system.

Introduction

Novice hackers, commonly referred to as "script-kiddies" [1,2], use automated scripts and tools to scan large numbers of systems for well-known vulnerabilities. These vulnerabilities, and explicit directions on how to exploit them to gain access to systems, are documented by many publicly accessible websites. Using this technique, even novice hackers are able to successfully compromise many systems with very little effort.

While script-kiddie techniques are sufficient to penetrate many poorly secured systems, successfully breaking into more secure organization takes time, effort, and planning. These more sophisticated attacks can generally be broken down into several distinct phases:

1. Footprinting: This involves identifying all systems and mapping the network topology of the target organization.
2. Fingerprinting: During this phase, information on host operating systems and available services is collected in order to determine potential areas of vulnerability.
3. Exploitation: This involves gaining access to the system through vulnerabilities identified in the fingerprinting process.
4. Covering Your Tracks: In this step, the attacker will sanitize log files and any other leftover artifacts of the attack and attempt to ensure that all evidence of the intrusion is eliminated.
5. Installing Backdoors: Once access has been achieved, additional software such as trojaned system binaries are installed to ensure that the attacker continues to have system access.

This paper focuses on these more advanced system attacks. Understanding the techniques commonly used by hackers, and making sure systems are protected from these types of attacks, is critical to establishing good system security. The techniques described in this paper are very similar to those used in black-box penetration testing. By understanding and performing the steps outlined in this paper, the security posture and security capabilities of your organization will be greatly improved.

While performing these types of tests against your organization is important part of creating a secure environment, there are also risks associated with this approach. Scanning systems and attempting to use exploits can have serious repercussions, up to and including the actual crashing of networks and systems. When performing these types of tests, it is important to make sure that appropriate contingency plans are in place.

The remainder of this paper is organized as follows: Section 2 discusses footprinting techniques useful for mapping network topologies; fingerprinting methods are presented in Section 3. A very brief discussion of exploitation methods can be found in Section 4, while methods for hiding evidence of intrusion are the topic of Section 5. Section 6 describes some common methods of installing backdoors on systems, and some conclusions are presented in Section 7. In addition to formal references, a list of relevant websites is provided at the end of this document.

Footprinting

The goal of footprinting, sometimes referred to as reconnaissance, is to map the Internet connections of a particular company to a sequence of domain names, IP address ranges, and specific Internet-attached system IP addresses. This information is then used to probe for vulnerable systems and potential access points in the fingerprinting and exploitation phases of an attack. Note that thorough footprinting is essential, as any overlooked domain name, IP address, or system could potentially be the weak link a hacker is seeking.

The remainder of this section is devoted to an overview of the steps typically involved in the footprinting phase of an attack.

Step 1: Refine the definition of the target based on subsidiaries and geographic location

This is most easily done by searching through an organization's web site and looking for information on locations and subsidiaries. Other possible sources for information include newspaper and magazine articles, USENET postings, and search engines. For publicly traded US companies, the Securities and Exchange Commission has a search engine available called Edgar (`http://www.sec.gov/edgar.shtml`) which can provide insights into a companies subsidiaries as well as recent acquisitions.

Step 2: Identify the domains associated with the target

Domains can be identified by using the *whois* database, either through the Unix *whois* command, or via one of the web interfaces to whois (e.g. website `http://www.networksolutions.com`). Keep in mind that most *whois* servers will only display the first 50 records found in a search.

In some cases, the whois search results may return several different domain names. These domain names may or may not be associated with real networks or machines- some domain names are merely place holders to protect future product names or intellectual property. Once a promising domain is identified, it can be researched in greater detail. Excerpts from the output of a *whois* search on a hypothetical domain are shown below.

```
# whois widget.com

Registrant:

      Team Widget (Widget2-DOM)
      145 Widget Street
      Widget, NH 00112
      US

      Domain Name: Widget.COM

      Administrative Contact, Technical Contact:
         Widget, Alex (WI2636) alex@WIDGET.COM
         Team Widget
         145 Widget Street
         Widget, NH 00112
         603.555.1212 (FAX) 603.555.1212

      Record expires on 27-Dec-2002.
      Record created on 27-Dec-1998.

      Database last updated on 31-Jul-2002 22:50:32 EDT.

      Domain servers in listed order:

      NS01.WIDGET.COM              66.92.10.166
```

```
NS02.WIDGET.COM                    65.102.10.43
```

From this output, we can see that a variety of information has been obtained from this single whois query. Specifically, close examination of these query results reveals information such as user account names, postal addresses (useful for determining if this domain is in the geographic area of interest), and the DNS server IP addresses.

It is not uncommon for a single person to be the administrative contact for many domains. Hence, a whois POC (Point of Contact) query on the administrative contact may provide additional useful information about other domains.

It is also possible to use the whois command to determine all email addresses for a specific domain:

```
# whois "@Widget.COM."@whois.arin.net

 Widget, Dave (WI676-ARIN) dave{\char64}widget.com 310-555-1212
 Widget, Emerson (WI374-ARIN) emerson{\char64}widget.com 781-555-1212
 Widget, Jeanne (WI3156-ARIN) jeanne{\char64}widget.com 408-555-1212
 Widget, Louann (WI263-ARIN) louann{\char64}widget.com 512-555-1212
 [...additional output deleted...]
```

The DNS server IP addresses are of particular interest, as we can easily determine if these IP addresses are owned by Widget.COM. In the example below, we see the address being queried is indeed owned by Widget.COM.

```
# whois -a 66.92.10.0

 Widget Network (NETBLK-Widget-5) WIDGET-5

             66.92.0.0 - 66.93.255.255
```

Step 3: Interrogate DNS servers to extract additional information about target domains

The Unix nslookup command is used to query DNS servers. This command provides information about the systems comprising the domain, including their function, names and IP addresses. One can perform nslookup queries on specific names, or simply request all the records for a domain. The following nslookup command sequence illustrates how one would request all records for Widget.COM, and save them to file; this operation is called a zone transfer.

```
# nslookup
Default Server: ns01.Widget.COM
Address: 192.168.1.200
 > server 66.92.10.166
 > set type=any
 > ls -d widget.com > /tmp/widget-zone.txt
 ###################
 Received 367 answers (367 records).
```

In the above example, the zone transfer was successful and downloaded 367 records.

The reader should refer to the DNS man pages for complete information on how to interpret the output of nslookup. Data which can be extracted from nslookup output may include details on specific systems and functions, user accounts, available services, and references to other systems.

Step 4: Evaluate the network topologies of the target organization

In this step, the collected network IP addresses of the target organization are used to map IP traffic flow. This is done by using the Unix traceroute command, which prints out the route packets follow to the specified internet host. By examining the output of traceroute commands run against systems owned by the target, it is possible to map the network connections and topology. Output from a traceroute command is shown below.

```
# traceroute widget.com
traceroute to widget.com (66.92.10.161), 30 hops max, 40 byte pkts
1 0   0   0 216.191.97.41 pos5-3.core1-mtl.bb.attcanada.ca
2 15  0   0 216.191.65.173 pos8-1.core2-tor.bb.attcanada.ca
3 16 0 16 216.191.65.243 srp2-0.gwy1-tor.bb.attcanada.ca
[...output deleted...]
```

It can sometimes be useful to cross-reference these results with information obtained from DNS servers, as machine names themselves (e.g. ¨gate1@widget.com) can sometimes be helpful when mapping networks and attempting to identify gateways.

Keep in mind that large organizations may have multiple paths into their environments, limitations on traffic flows (such as traceroute), and geographically distributed environments. Investigating all known addresses is important, as many organizations have their network devices inconsistently configured- what is tightly controlled on one IP address may be totally open on an other one.

Although some organizations restrict the use of traceroute on their external systems, this is typically implemented by denying the default traceroute protocols. By using a specific UDP port normally left unprotected, (e.g., DNS port 53) it may be possible to circumvent these restrictions; an example of such a traceroute command is shown below.

```
# traceroute -p 53 10.250.10.1
```

Fingerprinting

Much can be learned from analyzing the information systems make available over the network; this information includes lists of available services and their versions, the OS being used, and other system specifics which can be helpful to know when attempting to gain unauthorized access to a system. Fingerprinting, also called enumeration, involves scanning systems and recording this information for later use in determining which target systems have exploitable vulnerabilities.

Three of the most popular network scanners are

- `nmap` (`http://www.insecure.org/nmap`),
- `hping` (`http://www.hping.org`), and
- `fping` (`http://www.fping.com`).

As discussing the capabilities of either tool in detail could fill an entire book, only some aspects of these three tools will be mentioned here.

Whereas most tools will scan only one host at a time, fping is capable of scanning many hosts in parallel, and is therefore recommended for large sweeps. fping works by sending out a ping packet to a specific host, and then immediately moving on to the next host in the list. If a host replies within a set time period, it is considered reachable and removed from the list of hosts to scan. If a host doesn't reply to the first request, several additional packets will be sent until the retry limit is reached, at which the point the host will be considered unreachable.

Some common types of fingerprinting scans are described below.

Ping sweeps

Ping (ICMP ECHO REQUEST) sweeps are used to determine whether or not a host is alive. If a host does not respond to a ping, it may still be alive, but configured to not respond to the ping protocol.

Some security conscious sites do not permit ICMP messages or limit the type of ICMP messages which can pass through their networks. When scanning environments who block the ICMP messages normally used in ping sweeps (echo and echo-response) an alternative is needed. For these environments port scanning, using either UDP or TCP packets, is the best mechanism to determine if systems are alive. Both *hping* and *nmap* provide a wide selection of options which can help in these circumstances. Refer to the their man pages and documentation for more details on how to successfully scan systems in these types of environments.

Other ICMP-based scans

Sometimes a host configured not to respond to pings will still respond to other ICMP requests, such as an ICMP ADDRESS MASK REQUEST or an ICMP TIMESTAMP REQUEST. Any type of response from a system is enough to provide topology information to the hacker. ICMP Tools such as *icmpquery* (http://www.angio.net/security) and *icmpush* (http://hispahack.ccc.de/programas) send ICMP packets to hosts in order to gather additional information about a system. For example, to learn more about the network environment in which a system is configured, an ICMP type 17 message (ADDRESS MASK REQUEST) is useful. Knowing a system's subnet mask can be helpful when mapping a network environment, as it provides information on how network boundaries are configured.

OS identification scans

The goal of these scans is to determine the operating system in use by the target machine. These scans are normally implemented by sending malformed packets to a system, and examining the responses. Scanning tools such as nmap contain a database of expected responses, and can identify the OS based on the responses received.

TCP scans

TCP scans are used to probe for listening TCP ports or services which may have exploitable vulnerabilities. Normally, TCP sessions are established through the use of a 3-way handshake: the client sends a TCP SYN packets to the server; the server responds to the SYN packet with a SYN/ACK packet; on receipt of the SYN/ACK packet, the client replies with an ACK packet to the server. At this point, a normal TCP session has been established between the client and the server.

The different TCP scans implemented by nmap and other scanning tools can use the 3-way handshake while doing port scan, but this isn't always ideal for several reasons. Since a 3-way handshake involves completely setting up a TCP port on the server, it is easy for the server to log the IP address of the client attempting to access the port. Generating lots of log files and records of access attempts is obviously something that hackers try to avoid.

For this reason, nmap includes a variety of other methods which do not result in a TCP session being setup, and consequently are not usually logged. These scans are based in large part on the TCP behavior specified in RFC 793[3]. A few examples of TCP scans are given below; for more complete information on this topic, see [4].

- TCP SYN scans which don't complete the full three way TCP handshake. In this case, the client sends only a SYN packet to the target port. If the expected SYN/ACK is received, then the port is in a listening state, and the client doesn't respond. Inversely, if a RST/ACK is received then the port is not listening.
- TCP FIN scans don't use any part of the normal TCP 3-way handshake. Instead they send a FIN packet to a port. Normally a FIN packet is used by a client to close an already setup TCP connection. UNIX systems will typically return an RST for closed ports. If a packet other than RST is received, then that port may be open and listening, which indicates that some type of (potentially exploitable) service is running on the port.
- TCP XMAS scans send a packet containing the FIN, URG, and PUSH flags to a specific port. As in the FIN scan, most systems will respond with a RST for closed ports, and some other type of packet if the port is open.

UDP scans

UDP scans are used to determine if a UDP-based service is listening on a specific port. Using UDP packets is a bit more difficult than TCP, as the UDP protocol is stateless. Normally a UDP packet directed to a port on which no service is listening should generate a return ICMP error messages stating 'destination port unreachable'. However, if ICMP messages are filtered, or if a system is configured to not respond, then little information can be gathered through this method.

Exploitation

Footprinting and fingerprinting processes allow for the development of a list of hosts, and a profile of the services and operating system each host is running. We can then consult any of the publicly available vulnerabilities databases in order to determine what vulnerabilities exist on a given system, and for specific instructions on how to exploit them.

There are many publicly available sources of information on vulnerabilities. Mitre hosts the Common Vulnerability and Exposures (CVE) database (`http://cve.mitre.org`). Other organizations, such as AntiOnLine (`http://www.antionline.com`) and Internet Security Systems (ISS) (`http://www.iss.net`), host their own vulnerability databases. The bugtraq mailing list (`http://www.securityfocus.com`) is also an excellent resource as it is dedicated to the full disclosure of vulnerabilities.

It is also worth noting the existence of tools such as *nessus* (`http://www.nessus.org`), which can scan systems and determine if a particular vulnerability is present. Typically this type of tool will provide the CVE number for a vulnerability thereby making it very simple to look up and exploit the vulnerability.

As the actual exploitation of individual vulnerabilities was discussed in a previous paper [2] and many other Internet articles and security books[5], it is not discussed in any additional detail here.

Covering Your Tracks

Removing all traces of a compromise from a system requires that log files be purged of any error messages which may have been generated. Depending on which vulnerability was exploited, records of login attempts for a particular user account may also need to be cleared.

Two popular tools for cleaning up after a successful intrusion are *wipe* and *zap*, both of which are available from most security-related websites. *wipe* is used to remove log entries from lastlog, WTMP, and UTMP. *zap* cleans out the WTMP and UTMP login data for a a specific user.

More advanced hackers review the systems configuration to determine if log data (such as SYSLOG) was forwarded to a central repository. If so, the hacker will realize that their attack is more then likely going to be noticed,

and attempt to find a way to gain access to the centralized SYSLOG server in order remove any incriminating entries. This attacker may also flood the SYSLOG server with many bogus messages in an attempt to drown the real errors in a flood of noise apparently coming from a variety of sources.

Installing Backdoors

Once an attacker has covered up any evidence of their successful attack, a good hacker is going to make sure that they can always access this machine without being forced to exploit the vulnerability they used to initially gain access. In fact, some hackers will actually remove the vulnerability used to gain access from the system so that other hackers are not able to gain access through this means.

To ensure future access to a system, a hacker may use one or more of the following schemes:

- Start a new telnet or Secure Shell service on a typically unused port while making it appear to be something innocuous
- Replace the telnet or login binary with a trojaned, or maliciously modified, binary which will provide a root shell given a particular userid and password.
- Install an IRC (Internet Relay Chat) bot which will respond to instructions received from the attacker.
- Install remote administration software.

Anytime a system is found to have a known vulnerability, the system must be carefully reviewed to determine if it has been trojaned or modified in any way by an attacker. Several tools are available to assist with this process. Useful tools include TripWire, the Solaris FingerPrint Database[6], *lsof*, and netstat. Keep in mind, though, that an attack may also have trojaned system binaries such as *lsof* , *netstat* , *ps* , and *ls* to hide their backdoors. If there is any doubt about a system's integrity, either use binaries from a known secure system, or reinstall the OS from CD to return to a known system state.

Network sniffers are another type of tool frequently installed by hackers. Sniffers provide a mechanism through which an attacker can learn more about the network configuration, extract user ids and passwords from network traffic, and generally wreak havoc on a network. The most popular network sniffer at present is probably *dsniff* (`http://www.monkey.org/~dugsong/dsniff`).

Conclusion

New vulnerabilities and their exploit code are released on a frequent basis. Any service running on an Internet accessible system may be the vehicle of a successful attack. All services should be monitored with this in mind. A unsuccessful intrusion attempt performed yesterday does not guarantee that the service is still invulnerable today.

Keeping attackers from successfully compromising your organizations systems requires proactive security management, constant vigilance, and some amount of luck. Despite your best efforts, an attacker may still be able to find a means to gain access, or launch a DoS (denial of Service) attack against your organization. As a system administrator or IT security expert, you need to find every vulnerability and exposure, while an attacker need only find one[7].

References

1. D. Dion, "Script Kiddies and Packet Monkeys – The New Generation of Hackers", January 2001. `http://rr.sans.org/hackers/hackers_list.php`, January 2001
2. A. Noordergraaf, "How Hackers Do It: Their Tricks, Tools, and Techniques", BluePrints OnLine, May 2002. `http://www.sun.com/blueprints/0502/816-4816-10.pdf`,
3. "Transmission Control Protocol", RFC 793, USC/Information Sciences Institute, September 1981. `http://www.faqs.org/rfcs/rfc793.html`
4. Fyodor, "Remote OS detection via TCP/IP Stack Fingerprinting", June 2002. `http://www.insecure.org/nmap/nmap-fingerprinting-article.html`
5. S. McClure, J. Scambray, and G. Kurtz, "Hacking Exposed: Network Security Secrets and Solutions", McGraw-Hill, New York, N.Y., October 2001.

6. A. Noordergraaf, "Enterprise Security: Solaris Operating Environment", Prentice Hall, Upper Saddle River, N.J., June 2002.

7. G. Kurtz, C. Prosise, "Audits, Assessments, and Tests (Oh, My): Penetration Testing Exposed", Information Security, September 2002.
`http://www.infosecuritymag.com/articles/september00/features3.shtml`

Tools

- Dig DNS Query Tool
- Dsniff network sniffer
  `http://www.monkey.org/~dugsong/dsniff`
- FPing network scanner
  `http://www.fping.com`
- HPing network scanner
  `http://www.hping.org`
- ICMPush ICMP query tool
  `http://hispahack.ccc.de/programas`
- ICMPQuery ICMP query tool
  `http://www.angio.net/security`
- Nessus vulnerability scanner
  `http://www.nessus.org`
- NMap port scanner
  `http://www.insecure.org/nmap`
- TripWire Integrity Checker
  `http://www.tripwire.com`
- Solaris FingerPrint Tool
  `http://www.sun.com/blueprints/tools/fingerprint.html`
- Web interface to whois
  `http://www.networksolutions.com`

WebSites

- Antionline (hosts a vulnerabilities database)
  `http://www.antionline.com`
- Common Vulnerability and Exposures (CVE) database
  `http://cve.mitre.org`
- Internet Security Systems (hosts a vulnerabilities database)
  `http://www.iss.net`
- Mailing list for disclosure of vulnerabilities
  `http://www.securityfocus.com`
- Securities and Exchange Commission database
  `http://www.sec.gov/edgar.shtml`
- Network Tools website
  `http://www.network-tools.com`
- Incidents Threat Monitor website
  `http://www.incidents.org`

Digital Content and the Internet

Andrew McRae
Cisco Systems Australia
amcrae@cisco.com

Abstract

The deployment of high speed networking for research or commercial use is well established, with many applications either in use or being planned. This has driven much of the core Internet bandwidth growth, and remains the key factor for research networks such as Internet2, AARNet2 etc. However, consumer access to the Internet has largely been narrow-band (dial-up) in the past, limiting the range of applications that can be delivered over this limited bandwidth. Recently, with the widespread deployment of broadband technology, bandwidth upwards of 256kbps is available to homes via xDSL and cable modem, allowing the downloading of multimedia content including audio and video.

Future broadband technology is being planned that will offer much higher bandwidth, such as 10Mbps up to 1Gbps to the home, enabling delivery of real time high definition video content, and opening the way for new Internet applications targeted at a home market.

Consumer demands for multimedia content can be very high, as evidenced by the Napster phenomena, where a killer application suddenly emerged allowing sharing of MP3 audio files across the Internet. There were a number of outcomes:

- A huge increase in the amount of multimedia Internet traffic
- Recognition that such a multimedia application was driving broadband deployment of Internet access
- Eventual shutdown of Napster due to content copyright issues

The fallout from Napster continues to drive many of the issues in digital multimedia content delivery over the Internet. Technology now exists whereby digital content can be readily distributed over the Internet very cheaply. The barriers preventing cheap, high speed content delivery are no longer *technical* in nature, but have migrated into the realm of content control. Organisations such as the Motion Picture Association of America (MPAA) and the Recording Industry Association of America (RIAA) are working on a range of fronts to address concerns with protection of content.

This paper looks at these issues, with the recognition that technology has outstripped legal and content copyright issues, but that content protection is extremely important since the cost of creating such content is so high. A feature film may cost USD $100M to create, but in digital form can be copied many times or distributed over the

Internet very cheaply indeed, with no loss of quality. The content providers are rightfully loathe to allow such digital content to traverse the Internet, or to be transposed from other digital media into a downloadable form. It is clear that the primary driver for high speed home Internet access is this multimedia content, but how can content providers release their products over an Internet without mass pirating occurring, and loss of significant revenue? Legal redress alone will not prevent this happening.

One possible answer lies not in the legal or regulatory framework, but as a technical solution. Termed OCCAM (Open Conditional Content Access Management), it is a method of content delivery where the digital content is protected end-to-end, but provides for virtually any business model, even freely available digital content. This paper describes OCCAM, and presents it as a possible solution to the difficult problem of effective digital content provision over the Internet.

The Wired Home

There is tremendous interest in digital home networks for entertainment that allow the consumer to enjoy digital content anywhere in the home delivered on demand from the Internet or other networks. It is believed that over time

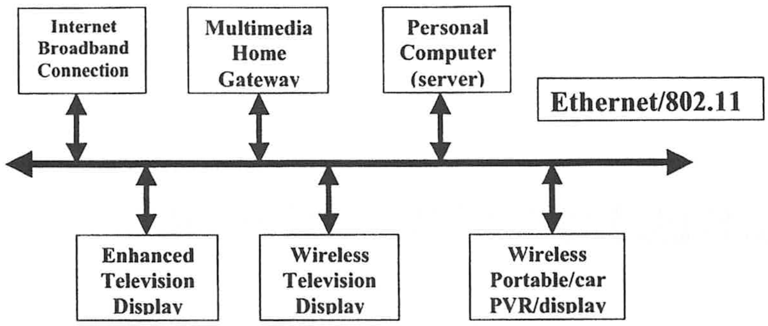


Figure 1: The wired home

content delivered over such next-generation digital home networks will replace traditional physical media such as DVD and VHS because of greater consumer convenience.

The home networks that are being built today are based on Internet technology, connecting the consumer to the Internet and enabling transferring of content from and to the Internet. While most existing Internet connections are narrow-band (dial-up), there are over 10M broadband connections in the US today, projected to grow to 30M connections by 2005. Broadband connections offer speeds of 384 Kbps or above and enable the downloading of multimedia content including audio and video. Future Internet broadband technology (Ethernet to the home) will offer speeds of 10 Mbps or greater to individual homes, enabling real-time streaming of high-quality video content.

One of the great advantages of the Internet Protocol is that it can run over a wide range of physical interfaces, including Ethernet, 802.11 wireless, HPNA, and others. These networking technologies represent large markets that are driven by large investments leading to tremendous cost-reduction. For example, the silicon cost of a 100 Mbit Ethernet interface is less than $1 today. The combination of large installed base, universal connectivity, transport layer independence, and low cost make the Internet Protocol an ideal platform for building the digital home network.

Figure 8 illustrates a digital home network based on Ethernet or wireless technologies, connected to the broadband Internet via cable, copper or fiber. A Multimedia Home Gateway provides content storage and imports content from traditional broadcast, satellite, or cable networks into the home network. Using the Ethernet or wireless home network, real-time or stored content is streamed from the multimedia home gateway to a multiplicity of network-enabled enhanced television displays and to portable devices such as PVRs in cars.

Digital Content

The NAPSTER experience highlighted how digital content can drive broadband usage. At one point, there were an estimated 77 million NAPSTER users, more than the total number of AOL subscribers.

With the introduction of the Compact Disc twenty years ago, consumers have enthusiastically embraced digital content as a way of providing accurate and consistent media delivery, and with the advent of readily available CD writing technology, consumers realised that digital content also provided a way of providing lossless replication of media, with no degradation of quality.

The development of the MPEG series of standards, especially MPEG3 (MP3) encoding of sound, provided a method of media encoding that was not only accurate enough for listening quality, but was compact enough so that fast and inexpensive delivery was possible over the Internet.

Content providers have recognised the potential of the Internet as means of content delivery for some time, and with the uptake of higher speed Broadband connections, it is now possible for a wider scope of content to be delivered i.e high quality video. However, content providers have also recognised the potential for illegal or unauthorised copying of content. Note that this is *not* a new phenomenon, since for hundreds of years copyright protection for written material has been available. So what makes digital content different from written material, and why aren't the current legal safeguards sufficient? For a number of reasons:

- The cost of reproduction of written material is higher than digital content. In the past, to cheaply reproduce a book required a significant upfront investment in typesetting and then mass production by printing presses. Even when copying techniques become widely available, the cost model was not attractive (generally, no one copies a book by photocopying it).
- The cost of creating the content is often a factor, and the subsequent price that is placed on the content. For example, creating a song or a music album is generally much cheaper than creating a movie, which can be upwards of $100M. This is often reflected in the price of the final product or the model of content delivery i.e music is sold on a CD and can be played by the purchaser as much as desired, whereas a movie is sold initially

as a one-time subscription (movie theatre ticket) and then as a short term rental (video rental) and then finally as a higher priced DVD or video. The model is designed to provide maximum return to the content provider.

- The ease of copying digital content and the cost of replicating and transporting it over higher speed Internet connections has meant that it is now possible to very cheaply and easily copy music, and to a lesser extent, video content. All that is required is that the content is provided in a purely digital form, so that lossless replication can be performed.

- Interestingly enough, the content providers seem to be less concerned about copying of *analogue* forms of media, though this has still been long recognised as a significant issue – the pirating of music and video content has been occurring for a long while, and continuing legal approaches will presumably stay in place to address this issue; a major difference is that analogue copying inherently reduces quality, and also requires some form of physical media of storage, limiting the impact compared to being able to store a series of bits on a hard drive.

So content providers and the powerful lobby groups that represent them have responded to this with various measures, both legal and technical. In effect, these measures are all designed to prevent the unauthorised copying of the digital content itself. One example is the recently enacted Digital Millennium Copyright Act (DMCA), which attempts to strengthen existing copyright measures by even outlawing the discussion or publication of any techniques that can be used to circumvent copy protection schemes. There has been much criticism of these acts, but there has to be an essential recognition that the rights of content providers must be protected. There is a strong tension here between content providers developing and lobbying for effective protection, and advocates who want to use the Internet for content delivery and maintain the traditional Internet values of openness, freely available content, and a strong antipathy towards censorship and restriction.

How can this dichotomy be resolved? There is a possibility that both sides will suffer; content providers may refuse to deliver digital content over the Internet, forcing the use of proprietary forms of communication (digital television etc.) without the openness and benefits of the Internet, maintaining the localised hegemony of content providers and media companies. On the other hand, digital content is seen as a major driver of Internet broadband bandwidth, and the use of the Internet as a delivery mechanism will lead to significant new applications – the lack of available digital content will stifle Internet growth, and splinter the methods available for content delivery.

As a manufacturer of networking equipment, Cisco has a vested interest in removing barriers to the growth of the Internet, especially when such a huge opportunity exists of providing high speed bandwidth to the home. Cisco also recognises that the needs of the content provider *must* be taken into account, so that an effective, workable and rational protection of the content can be achieved without relying on flawed legislation or compromising what are perceived to be the core values of the Internet.

OCCAM

One potential solution to this problem is OCCAM (Open Conditional Content Access Management), an end-to-end protocol for protecting valuable content from unauthorized copying, distribution and playback. This end-to-end approach to protecting content has enormous advantages, including:

- Digital content is never "in the clear" (i.e., it is always encrypted)
- Protects content during transmission across the delivery and the home networks
- Protects content while stored or cached in the public or home network
- Protects content over any interface, requiring no modifications to existing interfaces
- Does not rely on intermediate control points for content protection
- All content use remains under control of the content owner or content provider

OCCAM is separate from, but can coexist and interoperate with, any other conditional access, digital rights management, and content protection systems. OCCAM can import secure content from any proprietary system, including free over-the-air (OTA) transmission. In addition, OCCAM does not require, but can transparently deliver, content-embedded control information such as watermarks.

OCCAM solves the problem of content protection in a digital and networked world. Digital technology makes it extremely easy to store, copy and transmit copies of valuable content without the copyright holder's authorization. High value content such as a major feature film is extremely expensive to produce, easily costing tens of millions of dollars. Without end-to-end content protection, there is great risk that potential revenue is lost as some of the

audience receives copies for free through unauthorized channels. The development of the Internet and its evolution to ever-higher speeds significantly increases the risk of large-scale wide-area unauthorized sharing.

Using OCCAM, valuable digital content such as a feature film in MPEG-2 representation is encrypted by the content provider using the recently adopted Advanced Encryption Standard (AES). This encryption of the content uses a sequence of content keys, as is standard practice, with each key embedded in the content using an MPEG-2 entitlement control message (ECM). These ECM messages are encrypted using AES and a Secret Session Key (SSK).

The content owner makes only this encrypted form of the content available. This encrypted content may be stored on any digital storage device, transmitted across any network and duplicated at will. However the encrypted content can only be played on an OCCAM-certified player, with authorization from the content owner.

With OCCAM, a consumer requests authorization to play a feature film or other content from the content provider over a network connection, identifying his or her OCCAM-certified player. In response to such a request, the content provider generates a response message containing an authorizing ticket sent over the network to the OCCAM player, authorizing playback of the content, or else denying access. As part of this process, the content provider checks with a database maintained by the OCCAM industry association to verify that the identified device is certified and not revoked, and to determine the public key for the requesting player. The content provider then determines whether to authorize this playback or not, based on other information such as which region the requestor is in, and confirmation of suitable payment. The content provider then generates the encrypted ticket for the requesting consumer's OCCAM player containing the secret session key, encrypting this message with the public key of the player.

On receipt of the ticket from the content provider, the OCCAM player uses its private key to decrypt the ticket, determine the secret session key and is only then able to decrypt the content. Only the tamper-proof microchip in the OCCAM player with the private key corresponding to the public key is able to decrypt the authorizing ticket from the content provider to determine the secret session key. Therefore, only a specific certified OCCAM player is able to play the protected content, and then, only when so authorized by the content provider.

The OCCAM Secure Content Decoder

An OCCAM-certified player contains a tamper-proof microchip decoder that decrypts protected content and renders it for display, connecting directly to a display or through protected outputs. This microchip decoder includes a security certificate that specifies a public key/private key specific to this particular device, used for securely authorizing it to play back protected content. A certification authority certifies the public/private key. The secure content decoder is designed in such a way that digital clear content is not available outside of a tamper-proof microchip. Analogue video outputs (NTSC, PAL, and component video) are protected with an approved analogue protection system.

The diagram below shows the key building blocks of an OCCAM secure content decoder microchip. An important difference to traditional conditional access systems is the combination of the decryption and content decoding functions on a single, low-cost chip that exposes no digital content in the clear.

Acknowledgements

Much of this paper has been derived from OCCAM white papers and material authored by Andy Bechtolsheim and David Cheriton from Cisco Systems Inc.

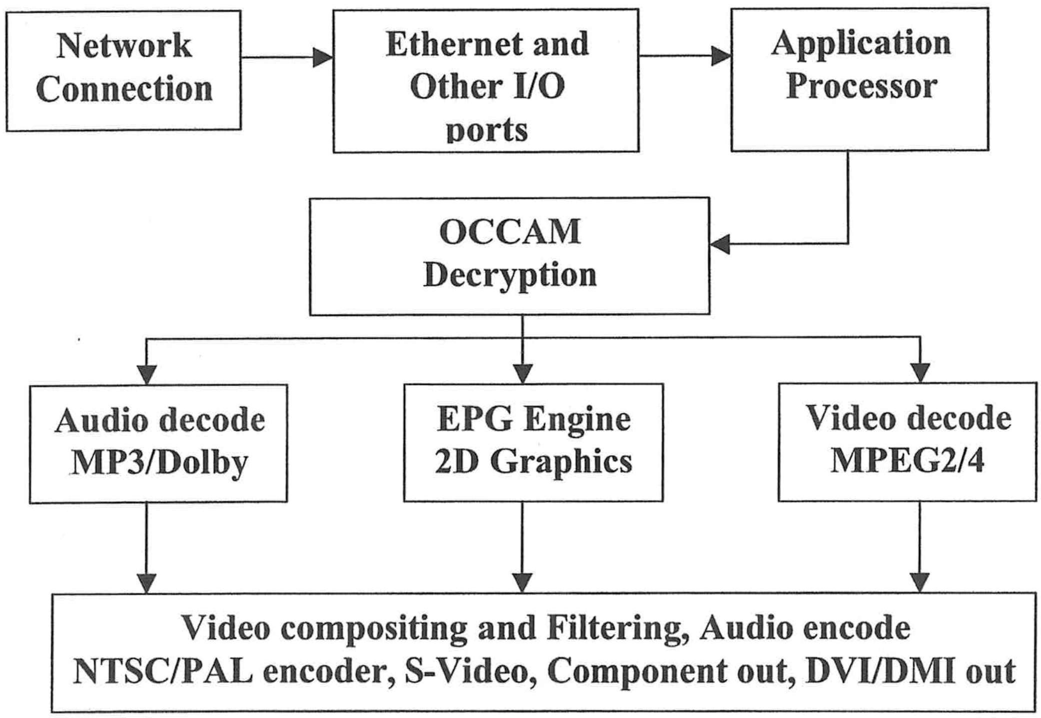


Figure 2: OCCAM building blocks

IPv6 in the Home Makes Sense

Aidan Williams
aidan.williams@motorola.com

Brian Haberman
haberman@innovationslab.net

Roger Kermode
roger.kermode@motorola.com

Abstract

Home networks today are predominantly comprised of PCs and almost exclusively run IPv4. There is, however, a trend away form the PC-only household towards a much richer set of devices and applications involving entertainment media, conferencing, and command and control that will see the rise of significantly more complex home networking scenarios. This paper aims to show that IPv6 will make such home networks possible and that the resulting IPv6-only home networks can be built today which support all the features of privately addressed IPv4 networks behind NATs. Further, this paper will show that IPv6 in the home allows applications to be easily deployed that have proved difficult in an IPv4/NAT environment.

Introduction

The number of IP capable devices in the home continues to gradually increasing over time. Only a few years ago, the state of the art in most homes was a single PC with a dialup PPP connection to the Internet. With the advent of broadband networks to the home, it is increasingly common for homes to have not just one but multiple PCs sharing a cable or DSL connection to the Internet. The market research firm IDC predicts that at least 30 million homes in the US will have multiple PCs by 2004, and an April 2001 survey by Parks and Associates showed that among 50% of broadband connected homes with multiple PCs have home networks.

In addition to PCs in the home, a new class of network-enabled appliances is emerging. Some examples are: networked digital audio players [MP3/RADIO] using the Internet to look up CD title and track information; personal video recorders [PVR] downloading new software, querying program guides and allowing remote programming; digital cameras that can send their images to your friends or to the printer [CAMERA]; and multiplayer games where players are in the house and on the internet.

The presence of broadband networks to the home is also changing the way that people work. Many companies, in order to attract and retain top talent, are dependent upon telecommuting. The growth of broadband access allows remote workers to efficiently access corporate resources. As home networks grow, so will telecommuting.

The needs of home networking

To a large degree, networking devices in the home can be considered a straightforward application of existing IP technologies. IP's ubiquitous nature creates great value in home networking since devices can communicate with services inside or outside the home, new services or old.

Simple Configuration

The home is different from the environments where IP is typically deployed today in one key way: the customer (the homeowner) is not a trained network administrator. Therefore home-based IP must be deployable by average consumers, people who generally know nothing of the mechanics of IP, routing, DNS, etc. Networks that can be

deployed in this manner are variously termed *zero-configuration* [ZEROCONF], *plug and play* [UPNP] or *auto-configuration* [IPv6SAC] networks. Whatever the term is called, it provides great benefit to anyone who has to sell, install, or extend a home network since the level of effort to achieve a working network is minimized.

The home is also very much an edge network. At present, ISPs typically provide residential users with a single IPv4 address, and levy charges for any additional addresses. Even when multiple addresses are provided, the installation and management of non-trivial home networks has been at best difficult since the suite of IPv4 protocols have not supported self-configuration (or zeroconf) operation well. Home users who wants to connect additional devices to the internet link usually install a Network Address Translator [NAT] in combination with a DHCP server which provides private IPv4 addresses to devices in the home [MINI-DHCP] according to static configuration burnt into it at the time of manufacture. The NAT maps these private addresses to the single global address supplied by the ISP when communication occurs with devices outside the home.

Simple Application Development

People buy do not buy networks for the sake of having them, they buy them to support applications and services. A well designed home network architecture should make it simple for applications to be developed that take advantage of IP connectivity between devices in the home, and devices in the wider Internet.

Historically, IP networks have used a simple communication model: applications just send data to a particular destination IP address and port number, and the network delivers it as best it can. Communication is assumed to be symmetric, and the IP address/port number pair is further assumed to represent the end-point. Today, there is a large installed base of applications, protocols, and APIs written under these assumptions. [INET-ARCH] and references describe the architectural principles of the Internet in more detail.

In the last few years, the increased deployment of privately addressed IPv4 networks [PRIVATE-V4] and Network Address Translators [NAT] has resulted in change to this communication model. NATs introduce a significant asymmetry in that it is now much harder (nigh on impossible) for applications *outside* a NAT to communicate with devices *behind* a NAT in the simple way described above. Communication initiated from behind the NAT to devices in the wider area Internet can proceed in the same straightforward way because a mapping between the private address realm and the public address realm can be automatically created. This asymmetry results from need to demultiplex packets arriving at the NAT for forwarding to the appropriate device behind the NAT. The *outside*-to-*behind* communication case requires the demultiplexing relationship to be set up manually, negotiated via a call setup signaling protocol such as RSIP [RSIP], or be supported in the application layer protocol being used.

Another observation is that IPv4 addresses by themselves can't be used to identify devices in the home because all homes using NATs are very likely to be using exactly the same private address space. A different identifier space is required which will allow the relevant NAT gateway to be discovered, and the device behind it to be identified. This adds complexity to a seemingly simple application like connecting to a web server embedded in one's home security system. Another approach is to standardize and deploy a signaling scheme between applications and NATs, and between NATs, as would be required for communication between two homes. This represents a significant architectural shift in the Internet.

The pros and cons of NATs are the subject of ongoing debate, references to additional information are provided at the end of this document [NATINFO].

The position adopted by this paper is that an ongoing requirement to deploy application specific code in NAT boxes to handle new protocols is the worst way forward and that a return to a unified address space through the deployment of IPv6 is a much better alternative. How this might be achieved for the home environment is the subject of later sections in this paper.

Some assumptions

- Many link technologies will be employed in the home with wide variations in bandwidth and delay: wireless, wired, power line, phone line, serial, water pipe, etc. IP is a good technology for connecting these different link types together.
- There will be at least *some* mobile devices in the home, perhaps a web tablet or a cell phone.
- Services inside the home will be accessed from outside the home. This could be accessing a security system, programming a VCR, conducting a videoconference, or home-to-home sharing of photos.

Service discovery and name service

Home networks need simple ways for locating services within the home and accessing them. In many cases, simply assigning a name to a service as one assigns a DNS name to a device may suffice. Other services may have large

numbers of options, which are better queried via a more general service discovery protocol like SLP [SLP]. Furthermore, some of these services will need to be exposed outside the home. Although the use of IPv6 does not imply fundamental changes to service discovery and name service protocols, changes are required to cope with longer addresses.

Security

Securing home networking is as important as putting locks on a home's windows and doors. If a home network is connected to the Internet, the homeowner needs simple and consistent ways of controlling access to their network, devices, and the information they contain in the same manner that physical measures prevent access to the home itself. The increasing use of wireless networks further complicates the security issue, by enabling access to the home network without having to be physically connected. For example, a neighbor may be able to listen to your network traffic, or a thief may be able to inventory your house without you being aware. Unfortunately, security schemes employed at the link layer vary widely and IP security is not easily configured. It would be beneficial if home network security were independent of, or at least allow unified configuration of each security system employed.

The most important feature of a security scheme is probably that it is ubiquitously available. If this is not the case, application developers will be forced to implement their own solutions, thereby resulting in a plethora of incompatible solutions that will mostly likely be hard to configure consistently.

What IPv6 Offers

While IPv6 is not a panacea to all the problems described above, we believe that it offers standardized solutions to many of them, and as such provides significantly better starting point for general purpose home-networking than IPv4. The main benefits afforded by IPv6 can be summarized as follows:

Large address space

Whereas IPv4 addresses are 32 bits in length and rapidly becoming short in supply, IPv6 addresses are 128 bits in length. IPv6 addresses are typically divided into two parts: 64 bits for routing, and 64 bits for a host identifier [IPv6ADDRS]. 128-bit addresses provide more than enough address space to allow assignment of globally unique IPv6 addresses to every device in the home.

Scoped addressing

IPv6 supports scoping: the ability to restrict the propagation of traffic to certain well-defined regions of the network [IPv6SCOPES]. Several well-known scopes are defined, link-local, site-local and global. Scoped address may be used not only to restrict the flow of traffic but also to restrict access to devices as follows. A device that wants to restrict its communication within a site (e.g. a home), may communicate using only site-scoped addresses. Likewise, devices using just link-local addresses can only communicate with hosts they are connected attached to. A clock radio may choose not to acquire global IPv6 addresses as a simple way of ensuring only householders can set the alarm and turn on the radio. Scoping is a helpful and simple way of controlling access to devices on a home network.

Auto-configuration

From the beginning, IPv6 has been designed with auto-configuration in mind. Hosts can automatically construct link-local addresses without configuration from a person or from a server in the network. Similarly, when routers are added to the network, they begin to advertise new network prefixes that hosts can use to construct site or global IPv6 addresses. IPv6 also allows hosts to have multiple addresses from any scope simultaneously bound to their network interfaces. Work continues in the IETF to fill in missing pieces like DNS server discovery [DNS-DISC].

Automatic tunneling techniques

Although IPv6 has a great deal to offer as a home networking solution, adoption by ISPs has been slow to non-existent. Fortunately, automatic tunneling techniques like 6to4 [6to4] allow home networks to deploy IPv6 even when providers do not offer native IPv6. A large IPv6 address space can be automatically constructed from a single global IPv4 address supplied by an ISP as follows:

The 32-bit IPv4 address is also the tunnel end-point, avoiding the need for manual configuration or an endpoint discovery protocol. 6to4 is appealing because it is deployable without requiring infrastructure support – thus allowing IPv6 to be driven from the places where it has benefit.

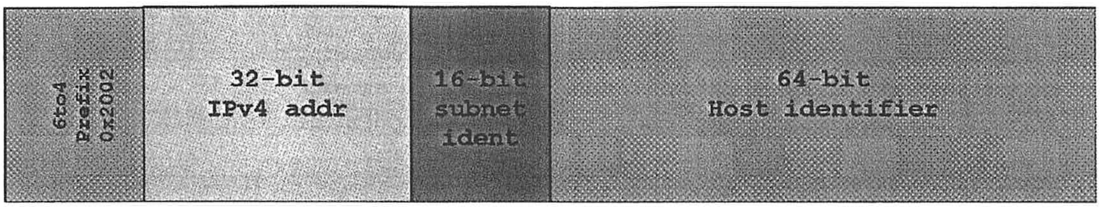


Figure 1: IPv6 6to4 address structure

Mandated security

The IP Security protocols [IPSec] are mandatory to implement in IPv6 stacks. This means that application designers can rely on standard security software being present on IPv6 hosts. The use of a firewall with IPv6 affords all the protection that a NAT can without the drawbacks.

A future path

A wide variety of protocols and applications in use today are adversely affected by the use of private IPv4 addressing and NATs [NAT-COMP]. Whilst IPv6 can be used immediately to avoid these problems, protocols that include IPv4 addresses in payloads will need revision before they will work over IPv6. This is often a straightforward change since no fundamental changes have occurred to the IP communication model.

Widespread deployment of IPv6 will only occur when IPv6 devices interoperate with the existing IPv4 Internet. The current techniques for achieving interoperability make use of NAT or application layer gateways (ALGs) to bridge IPv6 and IPv4 addressing domains, and inherit the problems of the IPv4 NAT/ALG solutions. It can be said that IPv6 backwards compatibility is no *worse* than what exists today with private IPv4 networks and NATs. On the other hand, pure IPv6 applications can avoid these problems entirely, now and for the foreseeable future.

Building a Pure IPv6 Home Network Today

It is the contention of this paper that IPv6 is already sufficiently mature for IPv6-only home networks to be constructed, which perform *as well as* the private IPv4 and NAT solution often used today. The scenario considered features a typical residential Internet connection supported by one globally routed IPv4 address. Within the home, only IPv6 is used. 6to4 is used to derive an IPv6 prefix for the home and for transporting IPv6 datagrams over the existing IPv4 Internet. Interoperation with the existing IPv4 Internet requires use of a protocol translating NAT [NAT-PT], a Transport Relay Translator [TRT] or application level proxies – as would be the case for a privately addressed and NATed IPv4 network.

Three services are needed to enable a pure IPv6 home network:

1. A 6to4 router providing IPv6 address space to the home, and transporting IPv6 datagrams over the IPv4 Internet. Connectivity to non-6to4 IPv6 devices can be achieved via a tunnel to a 6to4 router attached to the native IPv6 Internet.

2. A DNS ALG [DNS-ALG] running on a dual stack machine so that IPv4 address records can be translated into IPv6 address records usable by IPv6-only clients, and for following DNS referral chains that are likely to contain IPv4-only DNS servers. A variety of translation techniques embed the IPv4 address being contacted into the IPv6 address returned in the DNS response to the client. This allows the IPv4 address to be passed to the NAT-PT/TRT device, via the IPv6-only client.

3. A protocol translating NAT, or transport relay translator to support connectivity between the IPv6-only devices in the home network, and the existing IPv4 Internet.

The DNS proxy and NAT-PT (or TRT) interoperate as illustrated in figure 3:

The availability of IPv6 through the use of 6to4 allows applications to be easily deployed that have proved problematic in IPv4 networks using NATs. Three of these applications are shown in : home-to-home videoconferencing, control of a device from outside the home (the VCR), and access to information (the camera) inside the house.

These applications are difficult to deploy in privately addressed IPv4 networks using NAT since they require initiation of communication from *outside* the home.

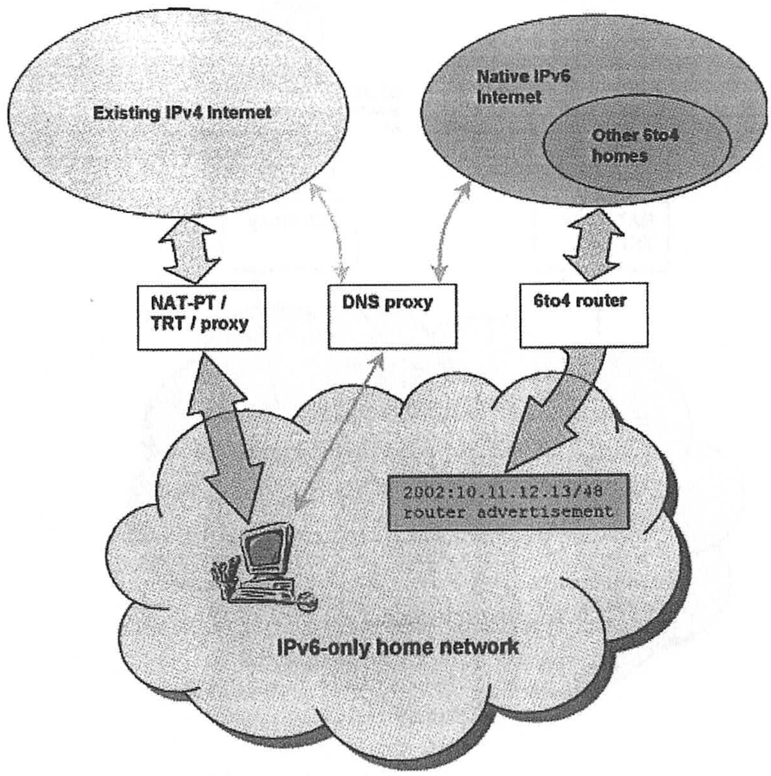


Figure 2: Services required for an IPv6-only home network

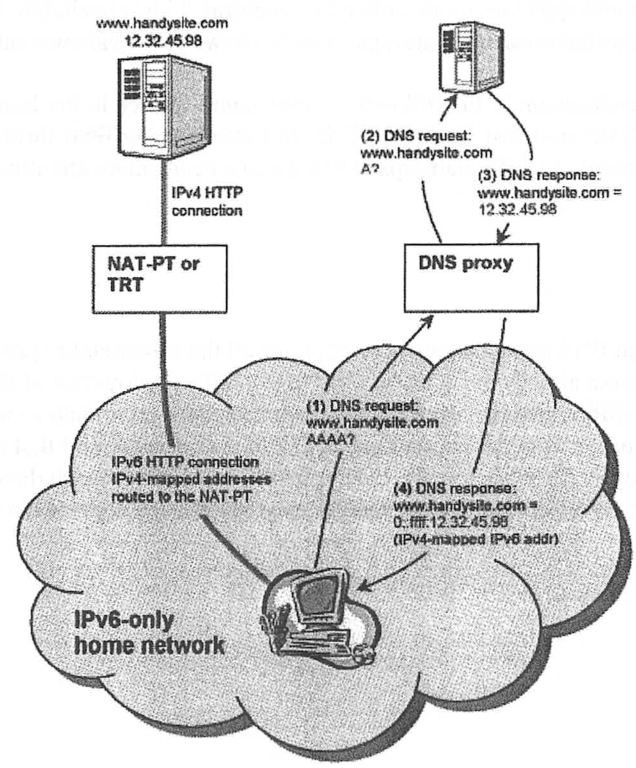


Figure 3: DNS proxy/ALG and NAT-PT interoperation

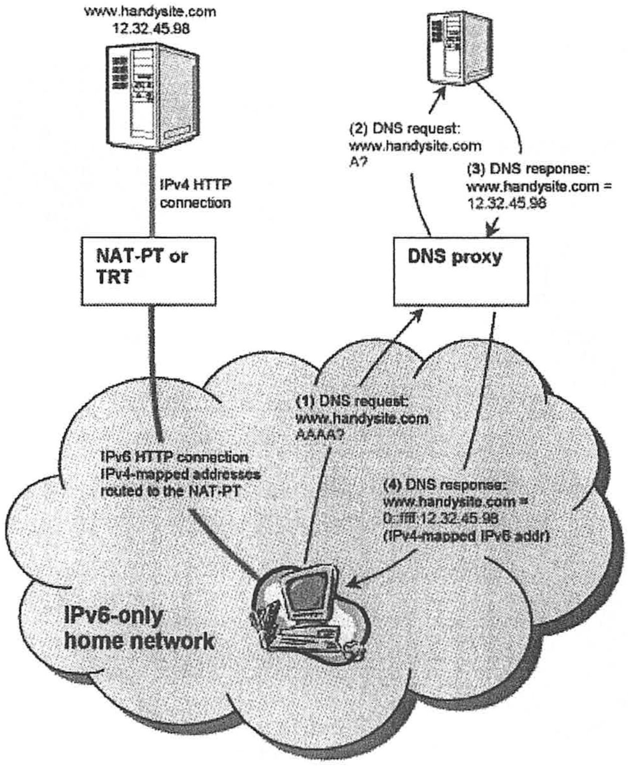


Figure 4: Applications enabled by IPv6 and 6to4

Missing pieces

Work on IPv6 has been progressing for many years, but as yet widespread adoption has not occurred. Support for IPv6 in popular operating systems and applications are only now becoming widely available. Key IPv6 specifications are now sufficiently mature that combinations of technologies can be shown to provide workable and useful solutions – for example, home networking.

Standards are under active development in the following areas: name service in the home using stateless DNS server discovery [DD-CHOICES] or multicast DNS [MDNS], and transporting IPv6 through IPv4 NATs [SHIP-WORM, STUN]. Simple management of a DNS namespace in the home needs more attention and is relevant to both IPv4 and IPv6 home networks.

Conclusion

The standards necessary to build an IPv6 home network that supports all the functionality provided by IPv4 privately addressed and NATed home networks already exist today. Furthermore, the deployment of IPv6 for home networks will ease application deployment difficulties by removing the communication asymmetry introduced by NATs. For these reasons we believe that the union of home networking and IPv6 makes sense and that as a consequence home networks will be a major early adopter for IPv6. Finally, the deployment of IPv6 through the use of 6to4 in the home will stimulate IPv6 deployment in the place where IPv6 has the most value – the edge of the network.

References

[6to4] B. Carpenter and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001. `http://www.ietf.org/rfc/rfc3056.txt`

[CAMERA] An Internet connected digital camera: `http://www.ricohzone.com/Product_RDCi700.html`

[DNS-ALG] P. Srisuresh et al, "DNS extensions to Network Address Translators (DNS_ALG)", RFC 2694, September 1999. `http://www.ietf.org/rfc/rfc2694.txt`

[DNS-DISC] D. Thaler and Jun-ichiro itojun Hagino, "IPv6 Stateless DNS Discovery", draft-ietf-ipngwg-dns-discovery-02.txt, July 2001, work-in-progress. `http://www.ietf.org/internet-drafts/draft-ietf-ipngwg-dns-discovery-02.txt`

[DD-CHOICES] D. Thaler, "Analysis of DNS Server Discovery Mechanisms for IPv6" draft-ietf-ipngwg-dns-discovery-analysis-00.txt, July 2001, work-in-progress. `http://www.ietf.org/internet-drafts/draft-ietf-ipngwg-dns-discovery-analysis-00.txt`

[INET-ARCH] B. Carpenter, "Architectural Principles of the Internet", RFC 1958, June 1996. `http://www.ietf.org/rfc/rfc1958.txt`

[IPSec] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998. `http://www.ietf.org/rfc/rfc2401.txt`

[IPv6SAC] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998. `http://www.ietf.org/rfc/rfc2462.txt`

[IPv6ADDRS] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998. `http://www.ietf.org/rfc/rfc2373.txt`
A new version of this document is underway:
`http://www.ietf.org/internet-drafts/draft-ietf-ipngwg-addr-arch-v3-06.txt`

[IPv6SCOPES] S. Deering et al, "IPv6 Scoped Address Architecture", draft-ietf-ipngwg-scoping-arch-02.txt, September 2001, work-in-progress. `http://www.ietf.org/internet-drafts/draft-ietf-ipngwg-scoping-arch-02.txt`

[MDNS] L. Esibov et al, "Multicast DNS", draft-ietf-dnsext-mdns-06.txt, October 2001, work-in-progress. `http://www.ietf.org/internet-drafts/draft-ietf-dnsext-mdns-06.txt`

[MINI-DHCP] B. Aboba, "The Mini-DHCP Server", draft-aboba-dhc-mini-04.txt, September 2001, work-in-progress. `http://www.ietf.org/internet-drafts/draft-aboba-dhc-mini-04.txt`

[MP3/RADIO] Many digital audio appliances on the market today (MP3 players, and internet radios) are network enabled. Some examples are:
`http://www.request.com/`,
`http://athome.compaq.com/showroom/static/ipaq/music_center.asp`,
`http://dec.hp-at-home.com/`

[NAT] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999. `http://www.ietf.org/rfc/rfc2663.txt`

[NAT-COMP] M. Holdrege et al, "Protocol Complications with the IP Network Address Translator", RFC 3027, January 2001. `http://www.ietf.org/rfc/rfc3027.txt`

[NATINFO-1] A variety of useful documents are linked from the IETF NAT working group charter web page: `http://www.ietf.org/html.charters/nat-charter.html`

[NATINFO-2] B. Carpenter, "Internet Transparency", RFC 2775, February 2000. `http://www.ietf.org/rfc/rfc2775.txt`

[NATINFO-3] T. Hain, "Architectural Implications of NAT", RFC 2993, November 2000. `http://www.ietf.org/rfc/rfc2993.txt`

[NATINFO-4] K. Moore, "Things that NATs break",
http://www.cs.utk.edu/~moore/what-nats-break.html

[NAT-PT] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", RFC 2766, February 2000. http://www.ietf.org/rfc/rfc2766.txt

[PRIVATE-V4] Y. Rekhter et al, "Address Allocation for Private Internets", RFC 1918, February 1996.
http://www.ietf.org/rfc/rfc1918.txt

[PVR] The SonicBlue ReplayTV 4000 is a Personal Video Recorder (PVR) requiring a home network.
http://www.replaytv.com/partners_products/replaytv.htm
See also http://www.allnetdevices.com/wired/opinions/2001/01/17/tivo_and.html.

[RSIP] M. Borella et al, "Realm Specific IP: Framework", RFC 3102, October 2001.
http://www.ietf.org/rfc/rfc3102.txt

[SHIPWORM] C. Huitema, "Shipworm: Tunneling IPv6 over UDP through NATs",
draft-ietf-ngtrans-shipworm-03.txt, October 2001, work-in-progress.
http://www.ietf.org/internet-drafts/draft-ietf-ngtrans-shipworm-03.txt

[SLP] E. Guttman et al, "Service Location Protocol, Version 2", RFC 2608, June 1999.
http://www.ietf.org/rfc/rfc2608.txt

[STUN] Rosenberg et al, "STUN - Simple Traversal of UDP Through NATs",
draft-rosenberg-midcom-stun-00.txt, October 2001.
http://www.ietf.org/internet-drafts/draft-rosenberg-midcom-stun-00.txt

[TRT] J. Hagino and K. Yamamoto, "An IPv6-to-IPv4 Transport Relay Translator", RFC 3142, June 2001.
http://www.ietf.org/rfc/rfc3142.txt

UPNP Universal Plug and Play: http://www.upnp.org/

[ZEROCONF] The IETF zeroconf working group charter has a definition of zeroconf:
http://www.ietf.org/html.charters/zeroconf-charter.html

Cfengine and FAI - Managing and Building 100s of Servers

BULLETPROOF NETWORKS

September 2002
http://www.inodes.org/papers/crcms

Acknowledgement

I would like to acknowledge my employer Bulletproof Networks, without whose help this work would not have been possible. Special thanks also goes Bel, Horms, Liam and Anthony for their valuable assistance.

The gods are just, and of our pleasant vices
Make instruments to plague us.

Shakespear on System Administration
(King Lear. Act v. Sc. 3.)

Introduction

"Aggh, I just wasted my whole day making config changes and upgrading ssh because of a security exploit!"

Once upon a time when I first started using Linux, I installed a copy of Slackware onto a 486 (my only machine at the time). If I wanted to install something I would typically visit SunSite and download the source. The next few hours would be spent compiling the software, fixing up any compilation or portability bugs and then finally issue *make install*, so that the software could finally be used.

Doing things in this manner was a valuable learning experience, but times have since changed, and frankly it would be difficult to find the time to do things in the same way.

Our jobs as systems administrators have become much easier. We now work in an environment where most Unix variants ship with different flavours of package management. Debian's *deb*[4] and RedHat's *rpm*[9] are the two most popular formats in the Linux Community. This has been extended even further through the use of tools such as *apt* and *autorpm*, which place packages at our fingertips and remove the need for Internet searches, etc.

Today I manage and am responsible for the administration of hundreds of servers, a large percentage of these being Bulletproof Networks Servers. These servers are mainly, managed customer servers which are spread all over Australia. Bulletproof Networks is currently one of the largest Managed Service Providers in Australia and as such manages business customers infrastructure.

When dealing with installation bases of this scale and a growth rate of up to 10 machines per month it becomes impractical if not impossible to maintain and deploy machines manually.

As an example of this, recently it was necessary to perform an ssh security update on all the servers under my management. Due to the CRCMS system still being under development, distributed upgrades where not possible. Simply installing a single package on all machines took 6 hours.

Therefore on and off for the past year as time permitted I have been designing a system based on *FAI* and *Cfengine* which dramatically reduces the time involved in deploying and managing servers and which will hopefully scale into the thousands.

Requirements

As a core part of its business Bulletproof Networks installs managed gateway servers for SMEs (Small to Medium Enterprises). These gateways consist of a Linux server which provides a redundant Internet connection, usually via ADSL and backup modem. These gateways also provide standard services such as NAT, firewall, email hosting, inter-office and roaming VPNs etc.

Initially these gateways where built manually using boot disks and Debian CDs. All configuration changes where done remotely via ssh and editing of the appropriate configuration files on the server. It became apparent fairly quickly that if the business was to grow rapidly and significantly then a different approach would need to be found.

A system needed to be found to automate most of these functions and to allow the system to scale into the thousands of servers.

The following requirements where devised:

- The building of servers, from blank hard disk to a functional system, needed to be totally automated such that minimal user intervention was required.
- The build process should work in such a way that the system could be installed by the PC suppliers at their premises. This would enable machines to be shipped directly to the customer.
- A centralised configuration repository was needed. This should be to the extent that an exact replica of a deployed customer machine, minus live data such as email, should be recoverable, in case of disk failure, with minimal effort.
- Any configuration changes for any machine should be made in the central repository and then pushed out to the required servers.
- Configuration version control.
- A system which would allow customers to perform simple administration tasks via the web, eg add users, change passwords. This needed to blend into the centralised configuration model.

Requirements Analysis

The build process

After some research it became apparent that most automated build systems, were based on an NFS-Root approach. The new machine would be booted either via a floppy or directly from the network, if supported by the hardware. A root file system would then be mounted via NFS which contained software which would then attempt to build the server.

In the case of replicator[2] this is achieved by simply partitioning the new disk and then using rsync[10] to replicate the parts of the build server's filesystem.

FAI[7] on the other hand is a bit more sophisticated in its method of software configuration. *FAI* has an understanding of the Debian[3] package management system and install process. It is therefore able to directly install required packages for each build.

FAI also supports the concept of classes which via a pre-configured file, a server could be placed in a different set of classes. This enabled different sets of software packages to be installed depending on the server's intended use. Thus there was some room to enable some pre-configuration of the servers to take place.

Another popular option is RedHat's[8] Kickstart functionality. This is somewhat similar to *FAI* in operation although no NFS-Root is used, simply two boot floppies. Bulletproof Networks had already standardised on Debian however, so this option was quickly ruled out.

In the end *FAI* was chosen due to its high flexibility. In hindsight this was a prudent choice as *FAI* has matured over time and has acquired new features that allow even further customisation and control of the build process.

Centralised Configuration

The choice of a system for the centralised configuration was eventually highly biased by the previous choice of *FAI*. *FAI* utilises *Cfengine*[1] to perform pre-configuration of servers based on the classes assigned to the servers. This seemed to be a system that would allow great flexibility in configuration of servers. Although, it has since moved away from this approach, now tending to rely on shell and perl scripts.

Cfengine also has the capability to distribute files to remote locations, at the time however this seemed to rely highly on NFS which was undesirable. Eventually CVS[5], using ssh as a transport, was chosen as a method of distributing files. The other inherent advantage of utilising CVS is version control, which was one of the base requirements.

FAI - Fully Automatic Install

FAI was written by Thomas Lange and according to the web page, *"FAI is a non interactive system to install a Debian GNU/Linux operating system on a PC cluster."*. FAI is definitely geared towards a Debian system but in its current form is extensible enough to be used to install any distribution with a mature package management system. There is also a version of FAI being directly worked on for Solaris which employs Sun's JumpStart technology.

Booting

FAI works by booting the new system into an NFS Root file system. Since the new machine has a blank hard disk it needs to be booted either via a floppy or through the network.

By far the simplest method and the one currently utilised is booting via floppy. An FAI boot floppy can be created by using **make-fai-bootfloppy**. This creates a `LILO` based floppy with either a kernel that supports DHCP or BOOTP booting. DHCP was chosen since it is easier to setup and the configuration is much simpler.

The machine being built needs to know its identity, ie some sort of host name or machine id. This is necessary so that the correct configuration can be downloaded from the central repository. This was initially achieved by setting a manual MAC address to host name mapping in the DHCP configuration. This became tedious as it was always necessary to work out the MAC address for each server and then modify the DHCP configuration (this was done using a web form). More recently this has been modified so that the install process itself prompts for the machine id. Although this does not allow a completely non-interactive installation it was easier for our purposes.

The alternative to booting from a floppy disk is to boot directly from the network. Since all servers are currently deployed with Intel EtherExpress Pro 100s, this can be simply achieved through the use of the PXE protocol supported by these cards. This significantly speeds up the build process. Details on setting up such a system can be found in a short paper by Simon Horman named *Disk-less Linux with the Intel Ether Express Pro*[6].

The LILO configuration on the floppy or the syslinux setup for network boots instructs the Linux kernel to mount the root file system via NFS rather than from a local disk.

The NFS Root

The NFS Root file system on the build server is created using **make-fai-nfsroot**. Using the values defined in **/etc/fai.conf**, a mini Debian installation is created in **/usr/share/fai/nfsroot**. Originally this was done by extracting the Debian **base.tgz** and then chrooting to install some extra packages. This is now done through the use of Debian's new **debootstrap** utility, which directly downloads the required packages from the Debian archive.

When booting off the NFS Root file system, booting takes place as normal. However the standard **/etc/init.d/rcS** script which normally begins executing standard run level scripts is replaced by the main FAI script which commences the build process.

It is highly recommended that a 100Mbps switch is used in the build environment. This significantly speeds up server builds. Especially when building multiple servers simultaneously.

Tasks and Hooks

The main build process is split up into a number of tasks, described briefly as follows.

- confdir - Starts up syslogd, configures the network via DHCP/BOOTP, mounts fai configuration at /fai
- setup - Configures some variables, starts sshd, retrieves local disk info
- defclass - Runs the class scripts to define classes
- defvar - Imports all the class variables based on the class definitions
- partition - Partitions all the disks
- mountdisks - Mounts all the disks
- extrbase - Installs the Debian base tar ball
- mirror - Mounts a local Debian mirror, if available
- updatebase - performs and `apt-get update` to bring the installation to the latest version
- instsoft - Installs any extra packages specified in the configuration
- configure - Executes all the configuration scripts
- finish - Cleans things up

One of the most useful features of FAI is its concept of **hooks**. For each of the tasks above a script can be placed in the hooks configuration directory. A hook can be used to supplement or replace any of the tasks listed above. The scripts are named as **task.CLASS_NAME**. For example the script **mountdisks.RAID** would be run if the RAID class was defined, just before the standard mountdisks code is executed. To totally replace a standard task the **skiptasks** command can be used inside a hook, usually at the end.

Here is an example of the hook Bulletproof Networks uses in place of the **extrbase** task.

```
# Mount the install image over NFS
mount $romountopt $buildhost:/export/base /base &&
             echo "/base mounted from $buildhost:/export/base"

# Copy it to the local disk
cd /base
find . | cpio -pm $FAI_ROOT

skiptask extrbase
```

This totally replaces the standard extrbase tasks, which would normally extract the standard Debian tar ball.

This method was used rather than the standard so that extra packages could be installed on the base system. This could normally be done using the instsoft task, however this became tedious due to some Debian packages which did not cleanly install and required manual intervention.

This situation should disappear as the reliability of Debian packages matures over time. Also newer Debian packages tend to do some initial configuration themselves by asking the user questions. There is some work being done on having the ability for the answers for these questions to be retrieved from a remote database. this would further simplify the build process.

Classes

The main advantage of FAI is its concept of classes. The *classes* directory has a set of executable scripts that are run during the defclass tasks. These scripts are used to define which classes a server is in.

This allows a great deal of flexibility in defining how a server is to be built. The best example of this is disk partitioning. Our servers come in two types: those with a single disk which are partitioned normally; and those with two disks. In the two disk case the disks need to be mirrored using RAID 1. During the defclass task the following script is run using the information gathered about the local disks in the setup task.

```
# two 20G+ Disks
($numdisks == 2) and
    disksize(hda,15000,70000) and
    disksize(hdc,15000,70000) and
    class("20GB-RAID") and
    class("RAID");

# one 20G+ Disk
($numdisks == 1) and
    disksize(hda,15000,70000) and
    class("20GB");
```

As can be seen above, depending on the physical layout of the machine, either the **20GB** or the **20GBRAID** class is defined. In the secondary case a different set of partitioning rules is used which partitions both disks identically ready for the RAID configuration. Later on due to the addition of the **RAID** class the actual RAID configuration using **raidtools** will be performed.

Variables

Once all the classes have been defined, any files in the classes directory **DEFINED_CLASS.var** will be sourced such that the variable definitions therein will be defined.

Partitioning and Mounting Disks

The partitioning of disks is based on configuration files placed in the **disk_config** directory. The files are named based on the classes defined in the classes task. Therefore following our example above, if a server with a single 20GB disk was being built then the file named **20GB** would be used.

```
disk_config hda
primary   /boot        30-128      rw
primary   swap         128
primary   /            2048        defaults,errors=remount-ro
logical   /var         4096        rw
logical   /tmp         512-1024    rw
logical   /home        4096        rw
logical   /export      0-          rw
```

This hierarchy is then mounted under `/tmp/target`.

The New File System

The **extrbase, updatebase and instsoft** tasks are used to build up the base file system. The extrbase task untars the Debian base.tgz on to the target file system. As mentioned earlier, on the Bulletproof Networks' system this has been replaced by copying a preinstalled base system across NFS. The new filesystem is then updated using `apt-get` in the **updatebase** task. During the **instsoft** task packages are installed by using each file in the `packages` configuration directory which has the name of a defined class. For example the DEFAULT package file looks like this:

```
PACKAGES install
cfengine debconf
ssh
bulletproof

# remove these packages
ppp- pppconfig- pcmcia-cs-
```

These packages can be installed from a local Debian mirror or straight from the Internet. The preferred option here is to install directly from the Internet but place a transparent HTTP proxy in the way. This ensures that the latest packages are always installed while minimising bandwidth consumption. It is probably necessary to increase the minimum size for objects that will be cached to disks to ensure packages are actually cached.

Configuration

Finally all the configuration scripts are run. These scripts all exist in the **scripts** configuration directory and are executed if the script name matches a defined class. These scripts can be used to perform some post install configuration for each server. For example the the following script called `BULLETPROOF` is executed for all our servers.

```
#!/bin/sh

# What is out clientid?
echo -n "Please enter host id: "
read hostid

mkdir $target/etc/bulletproof
echo $hostid > $target/etc/bulletproof/hostid

# Config repository dir
mkdir -p $target/export/bulletproof/cfengine
export CVSROOT=cvsuser@cvsserver:/export/cvs
export CVS_RSH=ssh
cd $target/etc
rm -rf cfengine
cvs co -d cfengine cfengine_config
cd $target/export/bulletproof/cfengine
cvs co -d local_config client_configs/$hostid
cvs co -d global_config global_config
cp $target/export/bulletproof/cfengine/local_config/etc/cfengine/cf.local \
    $target/etc/cfengine
```

This sets up the system ready to be configured by Cfengine after the first reboot.

Configuration Files

FAI also has a useful utility for copying config files to servers. In the `files` directory of the configuration a directory structure can be setup as follows.

```
/fai/files/
    etc/
       fstab/
          DEFAULT
          RAID
```

The *fcopy* command can then be used to copy files as follows.

```
fcopy /etc/fstab
```

If the RAID class is defined then `/fai/files/etc/fstab/RAID` will be copied to `/etc/fstab` otherwise the `DEFAULT` file will be used.

Cfengine

Cfengine was written by Mark Burgess. It is an automated configuration system. To quote from its web page[1],

> Cfengine, or the configuration engine is an autonomous agent and a middle to high level policy language for building expert systems which administrate and configure large computer networks. Cfengine uses the idea of classes and a primitive intelligence to define and automate the configuration and maintenance of system state, for small to huge configurations.

A class system similar to that seen in *FAI* is implemented by *Cfengine*.I suspect that this architecture formed a base for the FAI class system.

Cfengine operates on the idea of convergence, that is that it should take a machine in any state of configuration and move it closer to the actual configuration that has been defined. So, it should never make things worse, only better. This is a very desirable goal, especially when configuring large numbers of servers remotely. However it is fairly easy to achieve the exact opposite if care is not taken in crafting the Cfengine configuration correctly.

Methodology

In a similar vein to FAI, Cfengine's main primitive is classes. Rather than have multiple sets of configuration files one for every host, one common set of configuration files is used for all hosts. The actual configuration is then based on the classes that a particular host belongs to.

This model is based on the premise that in most organisations, configurations are remarkably similar across different hosts with only some changes being necessary based on slightly different installation environments. Cfengine also retains the capability for totally different configurations across hosts where necessary.

Features

At Bulletproof Networks Cfengine is mainly used for basic system configuration and other system administration tasks are left to other tools. However Cfengine can be used for a wide variety of other system administration tasks. The following feature list is from the product documentation:

- Check and configure the network
- Edit text files
- Maintain symbolic links
- Permissions and ownership of files
- Tidy (delete) the filesystem
- Automated mounting of NFS filesystems
- Checking for the presence of important files and filesystems
- Execution of user scripts and shell commands
- Process management

Classes

As mentioned previously Cfengine's main primitive is classes. In the same way as FAI classes define what is and isn't performed. Cfengine is more powerful since different actions can be taken based on sets of defined classes.

There are a set of default classes called Hard Classes. These are defined at runtime and cannot be added or removed.

- operating system architecture e.g. ultrix, sun4
- unqualified name of the host
- day of the week (Monday Tuesday..).
- hour of the day (Hr00, Hr01 ... Hr23).
- minutes in the hour (Min00, Min17 ... Min45).
- five minute interval in the hour (Min00_05, Min05_10 ... Min55_00)
- day of the month (Day1 ... Day31).
- month (January, February, ... December).
- year (Yr1997, Yr2001).

Other classes can be defined though the use of the *add_classes* directive within the configuration. These classes are referred to as soft classes. For example:

```
add_classes = ( BIND DHCP IPAC IPSEC MRTG SAMBA SNMP \
                SQUID SENDMAIL PPTPD VIRUS VIRUS_MCAFEE \
                ISP_PI LOC_SYD ADSL_INT ADSL_PPPOE
                ADSL_1500_512 BACKUP_MODEM MODEM_EXT )
```

The above example displays classes that would be added to the individual host configuration file for a Bulletproof Gateway. Most of the classes are self explanatory and are later used to determine how the server is configured.

The most basic example here are the LOC_SYD and ISP_PI. This means that the server is using a Pacific Internet service based in Sydney. These are then used to make configuration decisions for dial in numbers and name servers. This allows all these settings to be placed in one configuration file and used by all hosts depending on their configured classes.

It is also possible to run a script to add classes. This could add a RAID class in a similar way to the FAI setup.

Actions

Cfengine employs a set of actions to do all of its work. These actions include:

- mountall - mount filesystems in fstab
- mountinfo - scan mounted filesystems
- checktimezone - check timezone
- netconfig - check net interface config
- resolve - check resolver setup
- unmount - unmount any filesystems
- shellcommands - execute shell commands
- editfiles - edit files
- addmounts - add new filesystems to system
- directories - make any directories
- links - check and maintain links (single and child)
- simplelinks - check only single links (separate from childlinks)

- childlinks - check only childlinks (separate from singlelinks)
- mailcheck - check mailserver
- required - check required filesystems
- tidy - tidy files
- disable - disable files
- files - check file permissions
- copy - make a copy/image of a master file
- processes - signal / check processes
- module:name - execute a user-defined module

In the global configuration file an action sequence is defined which determines the order in which the actions will be performed. It is also possible to define temporary classes in the action sequence as follows. The temporary classes only exist while that particular action is being executed.

```
actionsequence = (
    shellcommands.install
    copy.standard.global
    copy.standard.local
    editfiles.standard
    shellcommands.standard
)
```

In this example, the copy action will be run twice. The first time it is run the **preinstall** and **global** classes will be defined and the second time the **local** class will be defined instead of the **global** class. This functionality is useful as it allows the same type of action to be executed multiple times but behave slightly differently depending on the defined class.

Example Cfengine script

The following example of a simple Cfengine script is used on our systems to setup the SNMP service.

```
copy:
  standard.global::
    $(global_repository)/etc/snmpd/snmpd.conf
      dest=/etc/snmpd/snmpd.conf
      owner=root group=root mode=0600

editfiles:
  standard::
    { /etc/snmpd/snmpd.conf
      ReplaceAll "BP_COMMUNITY" With "${snmp_community}"
    }

  standard.CUSTOM_SNMP
    { /etc/snmpd/snmpd.conf
      GotoLastLine
      InsertFile "${client_repository}/etc/snmpd/snmpd.conf"
    }

shellcommands:
  install::
    "/etc/cfengine/scripts/apt_check.sh snmp" useshell=false

  standard::
    "/etc/cfengine/scripts/check_diff.sh /etc/snmpd/snmpd.conf \
       /etc/init.s/snmpd restart"
```

If we follow the action sequence above, the first thing that is done is the *shellcommands.install* action. From the script above this means that we first look at the shell commands section. Since *shellcommands* is postfixed with the soft class *install* then only directives prefixed with the *install* class will be performed. In this case it is a custom *apt-check* script which checks if a package is installed and installs it if it isn't.

Next in the copy section an snmpd.conf is copied from the global repository. This is simply a directory full of preconfigured templates. The copy directive is very versatile and allows permissions and ownership to be modified among other things.

The editfiles action is performed next. This is the most versatile part of Cfengine. In this example we can see how it can be used to replace all occurences of one string in a file with a global variable. The second config entry in the editfiles sequence will only be performed if the class CUSTOM_SNMP is defined. If this is the case it allows some custom snmp directives to be appended to the configuration file. These are stored in a repository which is specific to this particular host.

Lastly the standard class part of the shellcomands config is performed. In this case the snmpd service will be restarted but only if the configuration file has changed. This is checked through the use of a custom script *check_diff.sh*. The initial versions of the file are saved away before any of the other Cfengine directives are run.

More examples

Due to space constraints please visit http://www.inodes.org/papers/crcms for more examples.

Configuration Files

In the CRCMS system, distribution of configuration files is handled by accessing a remote CVS repository via SSH. This is achieved through the use of public private key pairs. Each host has its own account on the configuration server from which it can obtain cvs access to three different repositories.

- cfengine_config - The Cfengine configuration files (Common to all hosts)
- global_config - The config templates (Common to all hosts)
- local_config - The configuration specific to this host

The local configuration is as minimal as possible and the configuration for most hosts consists of one configuration file which simply contains things like IP addresses, passwords and the configuration classes for the host.

Cfd

Cfd is the Cfengine daemon. It runs on the hosts and is used to start Cfengine. The configuration server connects to cfd and asks it to run Cfengine. This is all it can do. This ensures that all configuration changes are pull based. That is the configuration is pulled by the hosts from the configuration server via CVS and then Cfengine is run on the server. You can never actually push configurations to hosts.

Conclusion

FAI and Cfengine are extremely powerful tools, which can be used to build an impressive and extremely useful system administration system. While automation is the worthy goal of any system administrator, there are a number of things that must be kept in mind.

Building a system to do this is no easy task. You will probably start from scratch a couple of times. I've already done this twice and I am about to do so for a third time. As you gain more experience you will better understand how things work and tend to want to re-engineer things to improve the process.

Cfengine, while useful, can be an extremely dangerous tool. You need to ensure that any changes made to the configuration are carefully reviewed and follow change control process. At Bulletproof Networks this is done automatically by mailing CVS diffs on a commit to all the engineering staff. The problem with Cfengine is you can get it to do anything. Imagine if you changed the configuration to stop rather than restart sshd and cfd. This will hurt if you have thousands of servers.

Finally, always remember to not automate everything, coding your way out of a job is not the wisest thing to do.

Bibliography

[1] Mark Burgess. Cfengine, 2002. http://www.cfengine.org/.

[2] Sebastien Chaumat. Replicator, 2002. http://replicator.sourceforge.net/.

[3] Debian. Debian gnu/linux, 2002. http://www.debian.org/.

[4] Debian. Debian package format, 2002. http://www.debian.org/doc/FAQ/ch-pkg_basics.html#s-deb-format.

[5] Brian Berliner Dick Grune and Jeff Polk. Concurrent versions system, 2002. http://www.cvshome.org/.

[6] Simon (Horms) Horman. Disk-less linux with the intel ether express pro, 2002. http://verge.net.au/linux/diskless/.

[7] Thomas Lange. Fai - fully automatic install, 2002. http://www.informatik.uni-koeln.de/fai/.

[8] RedHat. Redhat linux, 2002. http://www.redhat.com.

[9] RedHat. Rpm package manager, 2002. http://www.rpm.org.

[10] Andrew Tridgell et al. Rsync, 2002. http://www.rsync.org/.

Big Data – Issues in Scalable File Serving

Michael A. Gigante

Introduction

The last few years has seen a significant revision in what is considered "large" in storage and file serving. In addition to larger and faster storage, the performance gap between CPUs, memory, buses and I/O devices has continued to widen, while the demand for faster, and larger I/O subsystems is growing.

Workflow issues are also now driving the storage architecture in many enterprises. With much larger numbers of files, and larger datasets, avoiding duplication and unneccessary movement of data has also become important.

The impact of each of these issues is considered and approaches to these problems are discussed. Finally, some examples of storage systems developed for large SGI customers are briefly described.

Storage Size

The capacity of individual disks has been approximately doubling every year. This has had an obvious impact on the average system storage capacity and the capacity of individual filesystems. Additionally, the total storage sold into the market each year has approximately doubled – supporting the notion of a modern "Data Explosion".

Traditional Unix filesystems such as Ext2 are no longer reasonable choices for enterprise-class storage. Limits in addressibility, and the ability of these older filesystems to maintain high throughput rates on real filesystems are both issues. However, it is the downtime necessary to recover from crashes or incorrect shutdowns that is the most serious issue. Without journalling or an equivalent technique, the recovery must be a complete internal consistency check of the entire filesystem. Large filesystems will therefore be offline for an unacceptable amount of time. Journalled filesystems such as SGI's XFS generally have recovery times of a few seconds.

While Network Attached Storage (NAS) devices are generally built on journalled filesystems, and have excellent IOPS/sec performance, they are limited in maximum bandwidth (as a direct result of using the LAN to provide access to the storage) and are confined to the small to medium storage size market (1Tb to 9 TB). Furthermore, one user accessing a very large dataset on an NAS will almost certainly saturate the LAN and thus reduce productivity of other users

Server Infrastructure

CPU performance has continued to grow at approximately 1.6X per year, memory capacity has been historically growing at approximately 1.3X per year and the gap between CPU speed and latency (memory access time) has continued to widen, from zero wait-state in 1986, to between 100 and 1000 instructions today. The result is that we are using deeper memory pipelines and larger caches. Furthermore, the ratio of memory capacity to I/O bus bandwidth has continued to worsen.

At the same, each new generation of disk has resulted in a 50% increase in capacity, a 30% increase in spindle bandwidth, and a 10% decrease in access time.

When all these effects are combined, we can see that from a CPU to disk latency of approximately $2x10^3$ instructions in 1986, we now see a latency of between $5x10^6 - 30x10^6$ instructions. We therefore should be prepared to spend a great deal of CPU effort to avoid or improve disk I/O.

This situation is not good for bandwidth eiter. With the use of multiple disk controllers and volume managers to stripe large numbers of spindles, storage bandwidth can easily scale to well beyond I/O bus and even system memory. This places limits on available storage bandwidth, despite a capacity to physically connect larger numbers of controllers and spindles.

Another trend in I/O infrastructure is the networking interconnect. Networks have been growing faster than busses and faster than spindle bandwidth for commodity networking. In the early 90s, Ethernet bandwidth represented approximately 5% to 10% of the ISA bus bandwidth. Now 1000baseT bandwidth approaches 100% of the dominant bus technology (66MHz pci-32). There is no current solution for 10GigE!

SGI has traditionally been extremely strong in the areas of I/O infrastructure. The NUMAFlex™architecture is based on low-latency, high bandwidth connections (NUMAlink™) between processing nodes called C-bricks. Each C-brick consists of 4 CPUs on a pair of local busses, local node memory and custom ASICS to interface to the NUMAlink™, support address translation and cache coherency. The bandwidth to the node's local memory is 3.2Gb/sec). The bandwidth of each NUMAlink™is 2x1.6Gb/sec (or 3.2Gb/sec full duplex), and the typical instruction restart latency is generally between 330 and 600ns (the restart latency is the time between when the cache miss is detected and the time the instruction is restarted. This includes CPU and refill overhead, but with no contention from other CPUs or back-to-back cache misses).

The latency varies because this is a cache-coherent Non-Uniform Memory Architecture (ccNUMA) and given its single global address space, a memory reference can be to a CPU's local node memory or memory on some other node. The nodes are connected via variations of a hypercube, depending on the total CPU count. As well as connections between C-Bricks, the same interconnect is used for connecting local storage devices (D-bricks), graphics devices (G-brick), PCI-64 devices (P-brick) and other I/O devices (I-brick).

This architecture, as well as being modular and scalable, also provides independent high-bandwidth connections between distributed main memory and I/O devices. This allows SGI servers to aggregate enormous bandwidth for I/O without encountering artificial bottlenecks. To put this in perspective, an Ultra3 SCSI controller can provide approximately 160Mb/sec of bandwidth, a 2Gb Fibrechannel device has a bandwidth of approximately 200Mb/sec and a PCI-64 bus operating at 66MHz has a peak bandwidth of 512Mb/sec while SGI's interconnect and memory bandwidth is 2Gbytes/sec.

Storage Architecture

Direct attached storage still dominates the market, although the rate of growth in Fibrechannel makes it likely that Fibrechannel will soon dominate the enterprise storage market. While SCSI is significantly cheaper at this point, 2Gbit FC over a switched LAN fabric is capable of 20% better bandwidth than an Ultra3 SCSI controller and provides other advantages such as storage consolidation and failover. Furthermore such SANs allow multiple hosts to access consolidated storage over a switched fabric, it enables true high performance shared filesystems.

The availability of SAN attached tape devices has significantly relieved a serious problem with backups. Since SAN attached tape units access the storage directly through the switched fabric, backups will not congest the LAN and will complete in greatly reduced timeframes.

Network Attached Storage (NAS) has had the highest growth rate in the past few years resulting from a combination of ease of installation, ease of administration (at least for small to medium storage requirements), and high performance for workload profiles that are metadata intensive. A NAS is likely to be unsuitable in the following scenarios:

- where high bandwidth is a requirement (since it uses the LAN),
- very large files (since the transfer time to the application server or workstation will be too long), and
- for very large filesystems (maximum filesystem size).

In Hierarchical Storage Management (HSM) systems, the storage is composed of 2 or more levels with different bandwidth and latency characteristics (e.g. high speed, low latency disk, and one or more levels of less expensive and greater capacity media). The HSM automatically manages the free space on the primary storage by migrating data to the bulk/nearline storage. A classic example would be a 2 level system comprised of a disk and a tape robot. Files in the HSM are online (the data associated with the inode is on the disk), or offline (the inode is online and marked as "migrated" and the data associated with the inode is stored on tape somewhere in the tape library). This is transparent to applications since if a file in the offline state is read, the I/O will block until the data is retrieved from tape. Latency and bandwidth can be managed by adjusting reserved free space, tuning migration policies, increasing the number of tape drives, the speed/capacity of the tapes, and the number of tape robots. HSMs are extremely effective for some classes of data and data access profiles. Some examples of effective sites will be discussed later.

File System Architecture

In this section, only journalled filesystems and related issues will be discussed. The simplest issues of scalability are design limitations in the filesystem. Such limits as the maximum file size, the maximum filesystem size, intrinsic parallelism, on-disk allocation strategy, read-ahead strategy, and the ability of the filesystem to transparently optimize physical disk I/Os given only standard application I/O calls.

As noted in the section above on Server Infrastructure, the long term trend identifies an increased relative latency between the CPU and the disk. Caching doesn't avoid the problem since the trend in storage capacity has been growing faster than the trend in memory capacity. Furthermore, applications and application data are growing at least as fast as the memory capacity. So in a mixed-use system we have proportionally less memory available for cache. Even in a dedicated file server, the adverse trends result in a lower memory cache/disk capacity ratio. Therefore, we need the filesystem to do some things better than we once would have been satisfied with.

Firstly, we must ensure that the filesystem is scalable in all aspects. Although XFS was first released in 1995, its design has proven admirably suited to the large I/O demands we see now. XFS can theoretically scale to 18 Petabytes, and individual files can be as large as 9 Petabytes. XFS has totally dynamic allocation of metadata – it has no theoretical limit on the number of files. XFS makes extensive use of B+ trees for internal, and on-disk data structures such as directory entries, file extents, and both allocated and free blocks in each allocation group. The filesystem itself is divided into allocation groups, each of which is a self-contained, contiguous portion of the filesystem. An operation on one allocation group is independent of every other allocation group. In large filesystems that are composed of a large number of spindles, this allows maximum concurrency for low latency and scalable bandwidth. The log device (for journalling) can be a separate device or can be internal to the filesystem. For large, high performance filesystems it is generally an external high-speed device.

Secondly, we can alleviate the latency issue to some degree by issuing larger I/O requests at a time. Therefore, I/O scheduling needs to be good. SGI's XFS is an extent-based filesystem, where an extent is a contiguous range of filesystem blocks. XFS extents can be up to 64Gb in size. Each file is composed of a set of extents and XFS attempts to minimize the number of extents for any given file. This is achieved by using delayed writes and by aggregating independent write system calls, i.e. coalescing adjacent writes to form a larger contiguous byte range to be written to disk. Note that the writes may be out of order or even random, and they will still be coalesced where possible. When the system schedules a physical I/O for a given in-core extent, a set of contiguous blocks are allocated on disk to store that extent. This on-disk allocation only happens when the data is being flushed to disk – i.e. we get the best possible result at the time we are forced out to disk. Provided there is sufficient free space in the filesystem, this results in a low degree of fragmentation and a favorable bandwidth to latency ratio for a given volume. Similarly, since the fragmentation is kept low, the ratio of reads to seeks is favorable, furthermore the coalesced buffers are available to the read-path as well, so XFS can minimize physical I/Os on the read path. It is also possible to pre-allocate contiguous space on the filesystem if it is of benefit to the application.

Thirdly, maintenance operations on extremely large filesystems should be possible while the filesystem is online. XFS has utilities for efficiently performing dump/restore on a live filesystem, for filesystem reorganization, and for growing a filesystem. As we will discuss later, the issue of backups is a critical one.

XFS has been proven to scale in practice, with a maximum demonstrated I/O bandwidth of 7.32Gb/sec on a single filesystem, and with a demonstrated 4Gb/sec on a single file. These demonstrations were on the previous generation of SGI ccNUMA machines (2000 series) which had NUMAlink™bandwidth of half of the current generation (3000 series). XFS filesystems are in production with tens of millions of files and in excess of 50Tb of online data in a single filesystem. Furthermore, in conjunction with SGI's HSM product (DMF) one production site has almost 900Tb of data under management.

While XFS has proven itself to be a formidable filesystem, it is not perfect and some work continues on identifying and fixing scalability and performance issues as they arise. One recent example is the change to synchronous log transactions and accompanying log I/O changes. LINUX XFS largely drove these changes for metadata-intensive workloads.

Administrative and Workflow Issues

General

As filesystems get larger, old practices become strained or break completely. One example is backups. The time to backup a multi-TB filesystem means that offline backups are not acceptable. Another is that if you backup utility has to go through the filesystem interface, performance will be unacceptable so other approaches are necessary. An example is SGI's bulkstat() system call which collects a complete inode group at a time directly through the on-disk structures. Also, that nightly script that compresses old data could be a bad idea when your file system get so large that the task can't complete in a single non-production shift.

Direct Attached and Network Attached Storage

XFS has been designed to allow very large filesystems. For other filesystems, an alternative is to use a number of smaller filesystems and make them available on different mount points. There are a number of reasons why a single filesystem can be a superior solution.

The first reason is global resource sharing. In a single filesystem, all users share the "headroom". In independent filesystems, productivity can be lost because the current filesystem has insufficient space even if there is plenty of free space scattered around other filesystems.

The second reason is administrative overhead; this is especially true where the filesystems are managed by some administrative GUI. If the maximum filesystem size is 1Tb and you have a total of 30Tb of data to manage, even an easy to use GUI quickly becomes a liability. Furthermore there may be additional overheads for managing backups and restores in the NAS environment since you have tape drives and tapes sets for each NAS (unless you want to backup over the LAN). Of course, backing up over the LAN is not a realistic proposition for filesystems larger than a few TB.

Where workflow issues dictate a pipeline of data from one process to another, copying the data from one filesystem to another can be a significant overhead. This is even worse if the filesystems are located on NAS appliances where the LAN is the only means of transferring data. This results in an unacceptable, adverse impact on the productivity of other LAN users.

Shared Filesystems on a SAN

Shared filesystems can solve these workflow issues by using a shared SAN fabric and consolidated storage. Unfortunately, many SAN installations fail to take advantage of this opportunity because the filesystem does not allowed coherent shared read-write access to a single filesystem from all the hosts attached to the SAN. In these cases, the storage is partitioned into different LUNs and access controls are used to allow only a subset of the storage to be visible to any given host. In this situation, the SAN is functionally equivalent to a local disk for each connected host.

In a truly shared SAN filesystem, the data pipeline can be a zero copy process, with no bulk movement of data across the LAN or across the SAN. In order to achieve a truly shared SAN filesystem, there must be filesystem and cache coherency across all the machines that access that filesystem. Once this is achieved, the shared filesystem can be used at close to local storage speed by each machine. SGI's CXFS (clustered XFS) provides exactly this facility. While metadata operations (including on-disk data structures such as the B+ trees, extents etc) are coordinated via a private network connection, all the data I/O is performed directly to the storage. For workloads that have reasonable metadata overheads, performance is just a couple of percentage points below raw XFS performance.

In early 2002, SGI made CXFS available on Solaris 8™and Windows NT™. CXFS will also become available on other architectures.

The administrative benefits of a large shared filesystem are very significant. There are no longer huge data transfers saturating the LAN, there is only a single working copy of data instead of local copies left behind at each stage in the workflow, centralized backups of a single filesystem operate directly on the SAN, and there are no network overheads at the start and end of each work process.

HSM

Hierarchical Storage management allows for automatic management of free space based on administrative criteria such as age of data, access times, modification times and size of the data. A HSM allows a site to provide expensive, high performance storage for the "working set" of active data, with the remainder of the data stored on secondary or tertiary storage, yet still accessible without any user or administrative intervention.

SGI's DMF™(Data Migration Facility) has a theoretical limit of 9 billion files, a maximum filesystem size of 18 million TB. In practice, numerous sites have 10's to 100's of TB under DMF management using a variety of storage media and DMF has been tested with up to 50 million files in a single filesystem. The largest site has close to a Petabyte of storage under DMF management.

The most common solution is to use tape libraries with a capacity of between 2 and 20Tb each. These libraries are capable of transfer rates of between 54Gb/Hr and 324Gb/hr. Individual access times for an offline file will typically be between 18 and 23 seconds. The total solution cost is well below that of disk-only storage, and the total data under management can be considerably larger. An appropriate migration strategies and a balanced mix of online and offline storage capacity can ensure productivity will be very close to disk-only storage for most workload profiles. There is also the advanta ge of having a high-performance tape subsystem that can be used for both backups and DMF.

There are some disadvantages to this approach, one is that where mechanical devices are involved, the administrative overhead is higher and while DMF will automatically determine when tapes need replacing, compaction etc, the administrator still has to do something about it! Furthermore, while the tape silo can be shared between DMF

and backups, the optimum performance criteria for drives/tapes are different for DMF and backup-only. Note that XFS dump and restore both recognize migrated files and intelligently deal with the online inodes and the offline data.

Example Installations

SARA

SARA (Stichting Academisch Rekencentrum Amsterdam) Is the Dutch national supercomputing center. The site is comprised of two 512 processor SGI 3800 series machines with one TB of memory and providing an aggregate of one Tflop/s of performance when delivered in 2000. For storage, there is 10TB of shared RAID5 filesystem (CXFS and DMF) over SAN and over 100TB of secondary storage in an existing StorageTek silo (managed by DMF).

There are 36 FC RAID5 controllers and a total of 360 36Gb 10000 rpm disks. The system provides greater than 2Gb/sec of aggregate bandwidth.

FNMOC

Fleet Numeric Meteorology and Oceanography Center, is the world leader in coupled air-ocean modelling, and provides worldwide weather forecasting for both the civil and defense sectors. This is a 200Gflop/s SGI 3800 series machine that generates approximately 1TB of data per 12 hour period. The center outputs 500,000 graphs, charts, analysis, forecasts and data sets per day, with 8TB of SAN storage (CXFS and DMF) with secondary storage of 80TB under DMF. This site also operates under SGI's Trusted Irix.

UCLA LONI

The UCLA Laboratory of Neuro Imaging has a CXFS cluster consisting of SGI 3000 series servers, SGI Onyx graphics supercomputers, and SGI Octane workstations (a total of 116 CPUs). There is a total of 6TB of online SAN storage (CXFS and DMF), and 40TB of secondary storage under DMF. The average dataset is approximately 20Gb, and the projected growth in LONI's imaging data (such as MRI, PET and OIS) is 8TB/year.

Conclusion

Large data poses a number of problems spanning the whole system, from computer architecture to filesystem design. SGI's systems, including XFS, CXFS and DMF have come from the big data domain of visualization and HPC but have proven to be suitable for many big data environments. The availability of CXFS for heterogeneous clusters represents a significant change in the big data landscape by bringing years of SGI's experience and expertise to new platforms and application domains.

SGI customers are driving the development of big data systems and the combination of off-the-shelf technology and our scalable hardware and software, they are able to meet their goals for high performance in the face of demanding I/O.

- Origin, NUMAflex, NUMAlink, Onyx, Octane, CXFS, XFS, DMF, are all trademarks of SGI.
- Linux is a trademerk of Linus Torvolds
- Solaris is a trademark of Sun Microsystems
- Windows NT is a trademark of Microsoft
- AIX is a trademark of IBM
- HP-UX is a trademark of Hewlett-Packard

Bibliography

[1] John Mashey
Big Data and the next Wave of Infrastress, Usenix 99
http://www.usenix.org/publications/library/proceedings/usenix99/invited_talks/mashey.pdf

[2] Steve Miller
I/O futures
SGI Network Engineer Technical Seminar, May 2002

[3] SGI Origin 3000 Series Owner's Guide (007-4240-002)
SGI Technical Publications

[4] Greg Astfalk
High-end Computing: What is Happening?
Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques

An Introduction to Quantum Computation & Communication

Rob Pike
Bell Labs
Lucent Technologies
Room 2C526
600 Mountain Ave
Murray Hill, NJ 07974
USA
`rob@plan9.bell-labs.com`

Abstract

How'd you like to solve an exponentially hard problem in polynomial time using infinitesimal energy? Think of the savings on laptop battery life alone. But is such a thing possible? In the bizarre world of quantum computation, theoretically yes. Quantum computation is more than just the use of very small things to compute. It exploits the fundamental and very odd properties of quantum-mechanical interaction to achieve profound parallelism, zero-energy calculations, unbreakable cryptographic codes, and other technological marvels. In interaction to achieve profound parallelism, zero-energy calculations, unbreakable cryptographic codes, and other technological marvels. In this talk I will discuss how the quantum world makes these things (theoretically) possible, the weird designs of quantum hardware and software, the current state of the race to build quantum devices, and what the prospects are for practical quantum computation in our lifetimes. The jury is still out, but the possibilities are mind-bending.

Friday 6th September 2002

The State of .au

Chris Disspain
CEO, auDA
`http://www.auda.org.au/`

The domain name system in Australia underwent a series of fundamental changes on 1 July 2002 when auDA, the Australian domain name governing body introduced a new regime including a competitive Registrar system and new domain name eligibility rules.

Chris Disspain is the CEO of auDA and in his presentation will outline the basis of the new regime, how its effectiveness will be reviewed and what pans there are for the future of domain names in Australia.

Who Moved My UNIX™?

John H Terpstra
Evangelist
Caldera International, Inc.

This plenary address will briefly review the history of UNIX, aspects of the Open Source movement, the birth and development of Linux, competitive elements in the IT market place, and trends that are shaping the future of IT. In particular this presentation will challenge attendees of this year's AUUG conference to take an active role in shaping the future of IT in their area of control.

The presentation will loosely follow the following outline:

Humble Beginnings

- University students drove market,
- Wrote unique applications
- Copied existing applications

What drove the market?

- Freedom
- Cost savings
- Control

World War III

- Hardware companies discover Unix
  - Hardware is not uniform
  - Support for hardware becomes platform specific
  - Ultimately Unix variant specific
  - Platform/vendor lock-in
- Software had to be custom ported to each platform
  - High cost of porting software
  - ISV's limited (locked in) to few hardare vendors
- Software companies made perilous choices
  - Software tied to hardware vendor's success / failure
  - Superceded hardware drove up customer costs
- Customers made dangerous choices
  - Hardware lock-in limits business solution scalability
  - Hardware vendors did go out of business

- Market Sensitisation
  - Everyone did it
  - Many still do!
- Open Source Software
  - Revolt against lock-out
  - Revolt against cost of ownserhip
  - Pursuit of new flexibility
  - Richard Stallman, GNU

Linux

- The Child of the Bubble
  - The Dot Gone Boom And BUST!
- Road to maturity
  - Unix comes Home to Linux
  - Unix source code open sourced
- User/Customer orientation
- UnitedLinux

The Real Enemy

- Form fact to fiction
- The Real Enemy

Information Technology Industry

- Show me the Money
- VARs and the market
- The fate of many VARs

Show me the Money

- Service == Results Orientation

About John Terpstra

As Caldera Evangelist, John H. Terpstra is responsible for Caldera's strategic initiatives involving Open Source publics, policies and projects. Terpstra connects key Open Source communities and Caldera business as well as product management and engineering. Terpstra, who has served Caldera as a vice president of technology, is a co-founder of the SAMBA project and a former TurboLinux executive.

At TurboLinux, Terpstra designed and managed the development of workstation and server products, trained VARs and negotiated sales deals with enterprise business partners. Prior to his position at TurboLinux, he served as the chief executive officer for Aquasoft Pty. Ltd., where he managed a global network of IT specialists and directed the training for Linux, UNIX and Windows NT technologies. Previous to Aquasoft, Terpstra served as national sales manager for InfoLink Computer Services and marketing manager for Houseman (Aust.) Pty. Ltd., a division of Portals, Ltd., UK.

Terpstra holds a certificate in Chemistry from the Queensland Institute of Technology and a graduate diploma in marketing from the NSW Institute of Technology in Australia.

Polythene PAM ain't what she used to be...

Martin Schwenke
IBM OzLabs Linux Technology Center
<*martins@au.ibm.com*> · <*martin@meltin.net*>

Abstract

This paper discusses the author's recent experiences with Pluggable Authentication Modules (PAM) under Linux, although most of the discussion applies to other PAM-enabled operating systems. Attempts to mix users defined in local files with users defined in an LDAP directory, and implement a defensive system administration policy, did not entirely succeed. The discussion covers the Name Service Switch (NSS) (and associated libc functionality), PAM, LDAP and user credentials, and concludes that some major changes are necessary to provide an authentication and credentials system that is reliable enough for mission critical systems.

Prelude: Polythene Pam

...
She's the kind of a girl that makes the "News of the World"
Yes, you could say she was attractively built
Yeah, yeah, yeah...
— Lennon/McCartney (© 1969 Northern Songs)

Introduction

The Pluggable Authentication Modules (PAM) framework was first suggested in around 1995 as a way of letting Unix system administrators configure a range of authentication technologies for a range of services, in a modular way. PAM was originally implemented on Solaris, though PAM support for Linux followed shortly afterwards. The original papers about PAM [2, 1] described an elegant system, which had the desirable feature that services could be considered orthogonal to authentication technologies.

Prior to this, with the introduction of Solaris 2.0 in the early 1990s, Sun had introduced another feature called the Name Service Switch (NSS) to support NIS+ [3]. NSS allows standard Unix user, group, host and other information to be gathered from arbitrarily specified back-end. The simplest such back-end is `files` comprising `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/hosts` and other standard Unix system files. Other, more elaborate back-ends, include LDAP directories, NIS+ maps and SQL databases.

For the most part PAM and NSS play together well, via the `pam_unix` module (described in more detail later), but when they don't the results can be confusing, if not totally unsatisfactory. Section "My Problem: Locally and Remotely Defined Users" provides an example of an attempt to implement a particular defensive system administration policy. The section "The Obvious Solution" outlines how this policy should be implemented, explains why it doesn't work and discusses several possible workarounds, each with their own problems. "Underlying Problems" discusses some underlying problems in and around PAM and some possible solutions are discussed in "Workarounds and Solutions".

Note this paper is not really about security. It is about being able to implement system administration policy.

My Problem: Locally and Remotely Defined Users

My computer system has two groups of users:

- those defined in `/etc/{passwd,shadow,group}`; and

- those defined in an LDAP directory.

I want members of both groups to be able to login to my system. I want these logins to be as reliable as possible.

The Risk

More generally, my system has two groups of users:

- those defined in local files; and

- those defined on a remote system.

Even more generally, my system has two groups of users:

- those defined in locals file, in a standard, well-known format, that is accessed by incredibly well tested code; and

- those defined in a third-party application, in a format that I don't really understand, accessed via several layers of networking software, from a remote system that is connected to my system via cables and various pieces of networking equipment.

If any of the vaguely defined components in the second point fails, the users defined in my LDAP directory can not login. That's a risk I have to take — I'm trading some reliability for the convenience of centralised administration.

Insurance Policy

Now consider that I *need* to login as `root` on my system to resolve an issue that is making my system unusable. `root` is a locally defined user and is the only user that really gives me full control over my system. To login as `root` there is no theoretical reason why I *need* to invoke any of the software or cables that deal with my remotely defined users. I don't want to introduce that level of unnecessary complexity to something as critical as logging in to my `root` account — it's not a risk I'm willing to take.

Therefore, it is my policy that none of the configuration for remotely defined users should be considered when I login as a locally defined user. I wish to configure my system so that, when a locally defined user logs in, the system exhibits the same behaviour as when there are no remotely defined users.

Why Am I Telling You This?

The type of configuration provided as an example with the `pam_ldap` package is shown below.

```
...
auth        sufficient   pam_ldap.so
auth        required     pam_unix.so use_first_pass
account     sufficient   pam_ldap.so
account     required     pam_unix.so
session     required     pam_unix.so
...
```

This configuration attempts to authenticate users and do account and session management via LDAP before looking in `/etc/{passwd,shadow}`. This violates my policy. Many `pam_ldap` proponents argue that:

- when the LDAP server is available, the `pam_ldap` steps take almost no time; and

- timeouts on the LDAP client-side can be adjusted so that when the LDAP server is unavailable the inconvenience can be minimised.

However, a bug in `pam_ldap` or the LDAP client libraries that causes a `SIGSEGV` will not allow the calling application to continue to authenticate any users, including the `root` user. This is one of the reasons why I have my policy.

The Obvious Solution

The obvious solution to the above problem is to put `pam_unix` before `pam_ldap` and make it `sufficient`, as follows:

```
...
auth        sufficient    pam_unix.so
auth        required      pam_ldap.so use_first_pass
account     sufficient    pam_unix.so
account     required      pam_ldap.so
session     required      pam_unix.so
...
```

This solution does not work. Moreover, with this configuration (and in fact *any* PAM configuration!) the policy set down in section "Insurance Policy" is still violated in an extremely subtle way.

It Doesn't Work?

One piece of the above configuration that doesn't work is the account management. There are two chunks of account management that need to be done:

- Password expiry checks, by looking at the various `shadow` fields.
- A host access check, done by comparing the hostname of the machine to the values stored in the `host` attribute of a user's LDAP `account` object.

The first of these is done by both `pam_unix` and `pam_ldap`, which seems to be an unnecessary duplication of work. This duplication probably stems from the recommendation that `pam_ldap` be listed first and be marked as `sufficient`. The second is done only by `pam_ldap`, which makes sense because `pam_unix` has no way of knowing about the `host` attribute.

If `pam_unix`'s account management decides that the given user's password has not expired, it returns `PAM_SUCCESS`. Since this step is marked `sufficient`, `pam_ldap`'s account management is never run. Therefore, the host access check is never done.

So, why does `pam_unix`'s account management succeed for LDAP users and what can be done about it?

What is a Unix User?

The search for enlightenment begins by asking a very simple question:

> What is a Unix user?

The answer to this is quite simple: a Unix user is one that is defined in a back-end that is available to NSS. For example, to use local files and then LDAP to lookup user and group information, the relevant entries in `/etc/nsswitch.conf` look like this:

```
passwd: files ldap
group:  files ldap
shadow: files ldap
```

For now, there are two things to note:

- NSS uses system-wide configuration. It can not be configured on a per-application basis. However, it is possible to customise the behaviour of certain back-ends.
- When searching for an entry given a user or group name, if the entry is found in `files` it is returned, and `ldap` is never consulted.

How Does PAM Handle Unix Users?

The deceptively simple answer here is: PAM handles Unix users via the `pam_unix` module, of course! This is true to an extent. `pam_unix` can handle any type of back-end from which NSS can be configured to *retrieve user information that includes the password field.* What's more, in certain contexts, `pam_unix` thinks that it can handle any type of back-end from which NSS can be configured to *retrieve user information.* That's quite a subtle distinction, especially since `pam_unix` doesn't check!

The first category includes standard Unix password systems, shadow password systems and NIS+. It does not include systems that can't or won't provide the password field, perhaps because they don't want to be quite so trusting. LDAP is one such system. To check if a username/password pair can be used to authenticate a user, the client must attempt to *bind* to an LDAP directory using the username/password pair. This operation can be performed without having to trust the client. However, in certain contexts, `pam_unix` thinks it can handle users in LDAP because passwords aren't relevant in those contexts.

So, this is why `pam_unix`'s account management succeeds for users defined in LDAP. `pam_unix` can see all of the relevant information via NSS, so it simply goes through the motions.

Workarounds

One possible workaround is to make `pam_unix` be `required`:

```
...
account     required     pam_unix.so
account     required     pam_ldap.so
...
```

This means that the `pam_ldap` account management is done for users defined in LDAP. However, it is also attempted for locally defined users, since PAM has no reason to stop after `pam_unix` succeeds. There are two problems with this:

- it violates the policy set down in "Insurance Policy"; and
- locally defined users will be unable to login since `pam_ldap`'s account management will fail.

Working Around the Workaround

Luckily, someone thought about this and organised a workaround in `pam_ldap`. It looks like this:

```
...
account     required     pam_unix.so
account     required     pam_ldap.so     ignore_unknown_user
...
```

This makes `pam_ldap`'s account management return `PAM_IGNORE` if the user is not defined in LDAP, which keeps PAM happy. What's more, this can be done without the module in question having to workaround the 'unknown user' situation. Welcome to PAM's 'pretty baroque stuff in square brackets':[1]

```
...
account   required      pam_unix.so
account   [default=die success=ok authinfo_unavail=ignore \
           user_unknown=ignore] pam_ldap.so
...
```

OK, that's much clearer! What? There's still a problem? Oh yeah! It still violates the policy set down in "Insurance Policy"...

A Working Workaround

Here is a workaround[2] that doesn't violate the policy set down in "Insurance Policy" (in any obvious way):

```
account     requisite     pam_unix.so
account     sufficient    pam_localuser.so
account     required      pam_ldap.so
```

`pam_localuser` is a non-standard module[3] that doesn't do any account management as such, but simply checks if the given user is locally defined via `/etc/passwd`. This allows PAM's account management processing to be short-circuited before `pam_ldap` is even considered. This appears to meet the desired policy.

[1] This is how Andrew Morgan, the Linux PAM maintainer, described this notation on the PAM mailing list [4] on 1997-08-04, nearly six months before it was actually introduced.

[2] This workaround appears to have been initially suggested by Nalin Dahyabhai <`nalin@redhat.com`> on the `pamldap` mailing list [5] on 2001-11-15. His version had `pam_unix` listed as `required`. Paul Hilchey <`hilchey@ucs.ubc.ca`> posted an updated version of the workaround on 2002-04-05, with `pam_unix` listed as `requisite`. This is a good improvement, since you really do expect that step to succeed.

[3] `pam_localuser` is distributed with Red Hat Linux, but is not yet part of the Linux PAM distribution.

One downside of this approach is that it may require two complete traversals of `/etc/passwd` (one by `pam_unix` and one by `pam_localuser`). If `/etc/passwd` is large, this might take a long time. Another downside is that it is illogical and counterintuitive — in the six months between 2001-11 and 2002-04 about half a dozen people asked[4] why their host access check wasn't working and needed to be told about the workaround. This doesn't include those who:

- still don't know they have a problem;
- didn't know they had a problem, but implemented the workaround when they saw it posted;
- knew they had a problem and saw the answer posted before they asked; or
- trawled through the mailing list archives to find the answer.

Underlying Problems

The previous section described an example problem and some workarounds. This section points out some underlying problems in PAM (or outside of PAM, as will soon be explained).

`pam_unix`

NSS is used to make user information available to the operating system and, therefore, defines the idea of a Unix user. `pam_unix` attempts to build on top of NSS and tries to cope with all of the users that NSS makes visible. However, "The Obvious Solution" has shown a case where it is necessary to know when a user is locally defined because the apparent generality provided by `pam_unix` is confusing. It is interesting that `pam_unix` tries to be all things for all users in a *pluggable* authentication system.

One service where `pam_unix` can not be all things to all users is the `password` service. `pam_unix` has built-in knowledge that allows it to change standard Unix passwords, shadow passwords and NIS+ passwords. `pam_unix` was obviously built with these three back-ends in mind and shows its limitations when mixed with other back-ends.

`initgroups(3)`

Even with careful configuration and judicious use of `pam_localuser`, a system that uses LDAP in NSS, will still need to timeout if the LDAP server is unavailable, and will crash if the client software contains a serious bug. This is because, after authenticating, an application that wishes to assume the full Unix credentials of the authenticated user must call `initgroups(3)` to initialise the supplementary group access list. `initgroups(3)` queries each back-end configured in NSS for a list of supplementary groups that contain the user.

This violates the policy stated in "Insurance Policy". However, this particular violation is very close to unavoidable.

Pretty Baroque Stuff in Square Brackets

The square brackets configuration notation, while perhaps a necessary evil in some circumstances, seems to promote configurations that are convenient but not necessarily good.

Workarounds and Solutions

`pam_unix`

There are two ways of making `pam_unix` more usable.

Split `pam_unix`

Recognise that at the end of the day `pam_unix` really knows about two different types of users: locally defined (`/etc/{passwd, shadow}`) and remotely defined (NIS+). Split `pam_unix` into `pam_files` and `pam_nisplus`, leaving two much simpler modules. This would help to make PAM more *pluggable*.

[4]On `pamldap` [5] and `pam-list` [4].

Further Complicate pam_unix

There are two options here:

- Add a localfiles option to pam_unix. When activated, this would make pam_unix check for the desired user in /etc/passwd at key points in its logic and return PAM_USER_UNKNOWN if the user can't be found.
- Make pam_unix able to cope with additional types of back-ends that may be configured in NSS. SuSE Linux's pam_unix2 takes this approach by using other modules like pam_ldap.

Further complicating pam_unix seems like a mistake, since some of its current problems stem from its complexity.

initgroups(3)

As noted in "initgroups(3)", PAM isn't responsible for setting up the supplementary group access list. PAM also isn't responsible for setting up the the userid or primary primary group membership. In general, PAM is not responsible for establishing a user's operating system credentials on behalf of an application. This is left to the application, using whatever means the operating system provides. This means that any application that wishes to establish Unix-like credentials must call functions like setgid(2), initgroups(3) and setuid(2) using information retrieved via NSS. NetBSD at least makes all of this easy by bundling all of these calls into a function called setusercontext(3) although, since this function is still called by the application, it still doesn't help to make credentials establishment any more pluggable.

The pam_group module allows an application to set extra groups via the pam_sm_setcred function. Other modules such as pam_krb use this function to set other back-end-specific credentials. However, pam_sm_setcred should not be used to set the basic operating system credentials.

Why don't PAM modules set operating system credentials, especially given the presence of pam_sm_setcred? The original PAM RFC [2] doesn't make this clear, but makes vague references to GSS-API — although GSS-API is a client/server authentication mechanism and clearly doesn't perform a credentials establishment role on current platforms that use PAM. The example code in the appendices of the PAM RFC performs setgid(2), initgroups(3) and setuid(2) calls.

On 1997-03-03, Andrew Morgan wrote on pamlist [4]:

> Credentials include things like (Kerberos) tickets. The natural extension of this is to make the setuid and initgroups calls part of this scheme, however Sun have ruled that these two things are actually in the domain of the application code.

Therefore, it looks to have been a decision made by Sun, the inventors of PAM. This decision has been regularly questioned on pamlist [4], but for now, it remains part of the 'standard'.

On 2002-06-27, Norbert Klasen suggested [5] the following workaround for the initgroups(3) problem when it involves nss_ldap:

> If local (system) users need not be members of groups held in ldap, one might introduce a "nss_min_uid" option [...] For example, if uid<100 then nss_ldap would skip the group lookup in ldap.

However, this was rejected by Luke Howard, author of pam_ldap and nss_ldap, who explained that initgroups(3) takes a username, not a uid as its argument, so it would still need to lookup the uid in LDAP.[5]

Workarounds aside, I think Sam Hartman, the Debian GNU/Linux PAM maintainer, summed up the situation quite well in his 2002-05-14 post to the PAM mailing list [4]:

> Long term, I think having PAM evolve to handle credentials establishment would be a net good; [...]
>
> Of course when you take things to their logical conclusion, PAM would be responsible both for the setuid call *and* initgroups; I think doing one without the other would be wrong.
>
> Getting to that ideal world would be very difficult; I think the PAM upstream, libc upstream and application writers would all disagree with us. We'd also need to think carefully about the API and potentially change things and better define things such that PAM could actually be responsible for user credential management. But hey if anyone ever wants to fight that battle, I'm certainly interested in helping.

[5] I must respond to Luke and ask whether the initgroups(3) implementation in nss_ldap would really go directly to LDAP to lookup the the uid for the given username . Surely the username should be looked up using getpwnam(3), but perhaps that isn't allowed inside the implementation of another NSS function?

Epilogue

Well you should see Polythene Pam
She's so good-looking but she looks like a man
Well you should see her in drag dressed in her polythene bag
Yes, you should see Polythene Pam
Yeah, yeah, yeah...

Get a dose of her in jackboots and kilt
She's killer-diller when she's dressed to the hilt
She's the kind of a girl that makes the "News of the World"
Yes, you could say she was attractively built
Yeah, yeah, yeah...
— Lennon/McCartney (© 1969 Northern Songs)

Conclusions

Despite the title of this paper, PAM hasn't changed very much, apart from that 'pretty baroque stuff in square brackets'. However, the world that PAM lives in has changed quite a bit, placing more varied demands on PAM. PAM has started to look a little left behind and dated. The biggest improvement would be to somehow integrate credentials establishment into PAM, so that too could be pluggable. It might be time for PAM to put away the polythene bag and try on the jackboots and kilt...

Thanks...

Many thanks to:

- Sam Hartman and Steve Langasek for useful discussions on `pam-list` [4].
- Stephen Rothwell and David Gibson for useful discussions about PAM, agreeing with some of the things I said and proofreading drafts of this paper.
- Melynda McDonald for being there while I wrote yet another paper.

Bibliography

[1] Vipin Samar and Charlie Lai.
Making Login Services Independent of Authentication Technologies.
Sunsoft, Inc.. An earlier version of this paper was presented at the 3rd ACM Conference on Computer and Communications Security, March, 1996.

[2] V. Samar and R. Schemers.
Unified Login with Pluggable Authentication Modules (PAM).
Open Software Foundation, Request For Comments: 86.0, October 1995.
`http://www.opengroup.org/tech/rfc/rfc86.0.html`

[3] Sun Microsystems.
Network Information Service Plus(NIS+): An Enterprise Naming Service.
1992.
`http://wwws.sun.com/software/whitepapers/wp-nisplus/`
`http://www.nrao.edu/computing/sol2/NISPlus-Admin-WP.ps`.

[4] PAM mailing list <`pam-list@redhat.com`>. Archived at
`https://listman.redhat.com/mailman/private/pam-list/` (list subscribers only).

[5] `pam_ldap` mailing list <`pamldap@padl.com`>. Archived at `http://www.netsys.com/pamldap/`.

Linux on the PowerPC 4xx

David Gibson
IBM Linux Technology Center

Introduction: PowerPC 4xx processors

The PowerPC 4xx series is a family of processors made by IBM for various embedded applications. As embedded chips, they are considerably less powerful and featureful than the chips found in desktop or workstation machines. However the 4xx chips are in the high-end for the embedded space and can run a fully fledged modern operating system such as Linux.

In particular, while the 4xx's MMU (a software loaded TLB) is simpler that that found in many desktop CPUs, it is sufficient to implement full virtual memory. Furthermore as the name implies the 4xx chips are an implementation of the PowerPC architecture, so they implement all the standard PowerPC instructions and most code written for a "normal" PowerPC will also run on the 4xx (albeit slowly). The 4xx chips, unlike desktop PowerPC chips, do lack a floating-point unit, but this is essentially the only difference between a 4xx chip and a "normal" PowerPC which can affect non-kernel (unprivileged) code.

The 4xx chips are designed for building single board computers. As such, in addition to the processor core, each 4xx chip includes many of the components needed for a fully working computer, including a interrupt controller, SDRAM controller and various peripheral devices built into the chip itself. Exactly what peripherals are present varies from chip to chip, but can include:

- Ethernet controllers (excluding the PHY)
- serial ports
- PCI host adaptors
- USB controllers
- IDE controllers
- real time clock
- ...

These on-chip peripherals are built as separate logic cores on the same piece of silicon. They connect to the processor core through several specialised on-chip busses. Thus, it is relatively easy to design a new chip with a particular set of peripheral devices by combining a CPU core with the relevant extra logic cores.

The 4xx chips also contain extra support for debugging embedded devices. This consists of a JTAG connection to which an external hardware debugger can be attached. Software on another machine can then use this hardware debugger to single step the processor, examine or change the contents of registers and memory, set breakpoints and perform other debugging tasks. It is even possible to step through low-level interrupt handlers and kernel boot code. This is particularly useful for an embedded device, which may well lack the normal IO devices (screen, keyboard etc.) which might be used for debugging on a more conventional computer.

An example: the 405GP

As a more concrete example, let's examine the 405GP, a reasonably typical chip from the 4xx family. This has a 200MHz PowerPC CPU core, with:

- a software-loaded TLB (see "The 405GP's MMU", below)
- 16kB of instruction cache and 8kB of data cache

- an interrupt controller (UIC)
- 4kB of on-chip memory (as fast as cache)
- an SDRAM controller
- an expansion bus controller
- a DMA controller
- a "decompression controller" (which allows code to be stored in a compressed form in memory and transparently decompressed when executed)
- a PCI host bridge
- an 10/100 Ethernet controller
- two serial UARTs
- an I^2C controller
- a number of general purpose IO lines (GPIO)

and (of course) the buses to hook all these together.

The 405GP's MMU

The MMU consists of a fully associative TLB with 64 entries, which are loaded by the operating system using the special `tlbwe` (TLB Write Entry) instruction. Each entry contains a virtual page number and a context number (known as the translation ID or TID), a real page number (a.k.a. page frame number), a page size, a "valid" (V) bit and a number of access control and attribute bits for the page. The TLB supports a number of different page sizes, varying from 1kB to 16MB, which can be used simultaneously (i.e. each TLB entry can have a different page size).

Power management

Minimising the amount of power consumed is clearly a desirable goal for many embedded applications, so 4xx chips have a number of power saving features. To start with the 405GP has only a moderate basic power consumption - it can run at full speed (200MHz) without a heatsink or fan. In addition many of the on-board peripherals and components can be switched off or placed in a low-power state when not in use.

Linux Support

Linux supports a number of boards based on 4xx processors, including the "Walnut", "Oak" and "Ebony" boards (IBM reference development boards for the 405GP, 403GCX and 440GP respectively), the EP405 (a PC/104+ form-factor, 405GP based board made by Embedded Planet), the Tivo PVR and others. This section looks at how we deal with the various peculiarities of the 4xx within Linux. Again, we'll use the 405GP as a concrete example.

Of course many "real" (as opposed to prototype or demonstration) embedded applications will be operating on a custom designed board. Hence for a particular application it usually won't be sufficient to just compile the kernel - some tweaking will probably be necessary for the particular hardware in use. Often custom boards are quite similar to reference design boards though, so the amount of core kernel work involved in such a port should be quite small.

As we've seen, the 4xx chips do not have floating point instructions. This causes no problem for the kernel code itself, since floating-point is never used there. For user programs the kernel can emulate a floating point unit, in which case Linux on the 4xx runs exactly the same user binaries as any PowerPC Linux machine. If floating point emulation is disabled then user executables and libraries must be compiled specially to use software floating point.

Although there is some code for 4xx support in the standard kernels from Linus Torvalds and Marcelo Tosatti, it is incomplete and largely broken. The most developed 4xx support current exists in the `linuxppc_2_4_devel` BitKeeper tree which, as the name suggests, is a tree based on Marcelo's 2.4 kernel maintained for PowerPC Linux development. A lot of the code for embedded machines in this tree has been contributed by MontaVista Software Inc.

Currently support for the 4xx in 2.5 is lagging behind `linuxppc_2_4_devel`. The code is gradually being merged into 2.5 though, and standard 2.5 is certainly much closer to having working 4xx support than standard 2.4.

Handling a software MMU

Supporting a software loaded TLB under Linux turns out to be conceptually quite simple. The kernel maintains a set of page tables in much the same way as on any Linux system. When a TLB miss exception occurs, a small piece of assembly code walks through the page tables, finds the appropriate entry and if valid loads it into the TLB (replacing existing entries in a round-robin fashion). If there isn't a valid entry we call the page fault handler just as we would on page fault generated by a hardware MMU. In order to keep the TLB fault handler as small and fast as possible the page table entries are laid out so that they can be loaded straight into the TLB with as little additional processing as possible.

As on all other PowerPC machines, the kernel uses a 4kB page size for normal page mappings. However the 4xx's support for larger page sizes is used for some optimisations. The `linuxppc_2_4_devel` kernel has a configuration option to enable "pinned" TLB entries. This option loads two 16MB TLB entries at boot time, which are never invalidated or overwritten. These entries cover the first 32MB of system RAM, including the kernel text and static data structures. Without this, entering the kernel (e.g. for a system call) tends to flush out most user entries from the TLB with entries for the kernel, thus reducing performance. This also significantly reduces the number of TLB misses taken while executing code in the kernel. It does however slightly reduce the number of TLB entries available to user code, which can slow down memory accesses when using a reasonably large working set.

There also exists code, although it hasn't been committed to the `linuxppc_2_4_devel` tree yet, which uses an improved technique to take advantage of large page TLB entries. This uses special "large page" entries in the Linux page tables. Linux on PowerPC normally uses a two-level page table, but a "large page" entry is a specially marked entry in the top level page directory which directly describes a mapping of a 4MB region (the area normally covered by all the PTEs in a second level table). Mappings of 16MB regions can also be described by setting 4 consecutive page directory entries.

At boot time we set up large page entries to describe a mapping of physical RAM (again including the kernel text and most of its data structures). These entries can then be loaded directly into the TLB (using large page TLB entries) by special code in the TLB miss handler. This again reduces the TLB usage by the kernel, but in a more flexible way which doesn't preclude user code from using all 64 TLB entries when appropriate.

Cache coherency issues

4xx chips, unlike most PowerPC CPUs, don't do cache-coherent DMA. That is to say the data cache in 4xx chips doesn't snoop the bus for DMA transfers and update or invalidate itself appropriately. This means that if DMA transfers are occurring then the CPU's view of memory (i.e. what's in the data cache) and a peripheral device's view of memory can get out of sync. This means some special techniques are required when using DMA.

There are two approaches to handling this situation. The first is to specially allocate some memory for DMA and map it marked non-cacheable. With the cache disabled the CPU's and the device's view of the memory will be consistent and there are no further problems. This approach is simple, but disabling caching can introduce performance problems, and this can be inconvenient if a driver is handed blocks of data which have been allocated by other parts of the kernel (e.g. network socket buffers) - using this approach would require copying the data into a new, non-cacheable buffer.

The second approach is to use normal cacheable memory, but to perform explicit cache writebacks and invalidates where necessary to ensure consistency. Using this approach at any given time a block of memory will "belong" either to the CPU or to the device. When the buffer belongs to the CPU there must be no DMA active in to or out of it, and when it belongs to the device the CPU must not read or write the memory (i.e. it must not pull any of the buffer into cache). A driver can transfer ownership of the buffer by flushing the cache[1]. This approach does require that no DMA buffer share a cacheline with any other kernel data structure.

Which approach is the most suitable depends on the nature of the hardware, and sometimes it's desirable to use both in the same driver. For example, consider the driver for the 405GP's built-in Ethernet controller. The hardware uses tables of descriptors to describe buffers to transmit from or receive into. At initialisation time, the driver allocates some uncached memory for these tables - because the hardware can write or alter descriptors in the tables at essentially any time, it's impossible to use explicit cache flushes to manage them.

At initialisation the driver also prepares a pool of buffers to be used for receiving packets. It does this by allocating memory (with `dev_alloc_skb()`), then invalidating the cachelines in each buffer, so that the buffers now belong to the device. It then informs the device of the locations of the buffers by writing descriptors into the Rx descriptor table. When a packet is received by the hardware the following will happen:

1. The device receives the packet and DMAs the data into the next available Rx buffer.

2. The device alters the Rx descriptor table to indicate which buffer it has used and signals an interrupt.

[1] Whether a writeback and invalidate or both is needed depends on whether the DMA is from memory, to memory or in both directions.

3. The driver's interrupt handler parses the Rx descriptor table to find which buffers have been filled.

4. The driver reads the buffer, processes the packet and passes it to the network layer. Reading the buffer pulls it into cache, thus transferring "ownership" back to the CPU - this is safe because the device is now finished with the buffer.

5. If the pool of buffers is low, the driver prepares more buffers as at initialisation time.

A somewhat similar procedure is needed for transmitting packets (outgoing DMA). Once the driver has been passed a packet buffer by the network layer the following will happen:

1. The driver does a cache writeback on the cachelines the buffer occupies (the device now "owns" the buffer).

2. The driver writes a descriptor for the buffer to the Tx descriptor table.

3. The device reads the new descriptor and transmits the packet, DMAing from the buffer.

4. The device alters the descriptor to indicate that it is done and signals an interrupt.

5. The driver determines which buffers have been handled by the hardware and frees them.

The 4xx chips are not the only non-cache-coherent CPUs which Linux supports, so there already exists some infrastructure for handling this situation. For the first approach, the function `pci_alloc_consistent()` is used by PCI drivers to allocate memory which is DMA consistent - this will be ordinary memory on cache-coherent processors and non-cacheable memory on non-cache-coherent processors such as the 4xx. For the second approach, there are a variety of functions such as `pci_map_single()` and `pci_map_page()` which are used to perform the necessary cache operations[2]. On 4xx, these functions are implemented in terms of the architecture specific functions `consistent_alloc()` (for `pci_alloc_consistent()`) and `consistent_sync()` (for the `pci_map_*()` functions), which are also used by non-PCI drivers such as those for the 4xx's on-chip peripherals.

On chip peripherals

Linux drivers have been written for many of the 4xx on-chip peripherals. For example, on the 405GP, the PCI bridge, Ethernet controller, UARTs, I^{2}C controller and GPIO lines all have drivers. There is also a driver for the on-chip IDE controller which appears on some other 4xx chips.

Since the on-chip peripherals are built as independent logic cores, a number of 4xx chips may share an identical peripheral. So, it's obviously desirable to share a common driver for the device between these chips. However the on-chip buses to which the peripherals are attached do not support any type of scanning or device detection - the kernel has to "just know" what devices are there.

Because of this, a kernel must be built to run on a particular 4xx chip. Specifying the chip at compile time will (amongst other things) include an object into the build which contains a table listing the various on-chip peripherals, their addresses, interrupt assignments and so forth. This table is used to provide information to the device drivers for the on-chip peripherals, so that they can locate their devices. In general the drivers won't need to directly know the chip they're running on - this makes supporting new 4xx chips with similar on-chip peripherals much easier.

The on-chip devices and their drivers are connected together by some code known as the OCP (for On-Chip Peripheral) subsystem. This keeps track of what devices are present and which drivers are active.

Board and initialisation issues

In addition to the devices on the 4xx chip itself, each board a 4xx chip is used on usually has some special devices itself (if nothing else boards frequently include an FPGA or CPLD with some control registers). The firmware, if any, on these boards is usually limited to some basic initialisation and loading, and also provides no support for hardware detection.

Hence, a kernel must generally be built for a specific board, as well as for a specific chip. Specifying a particular board in the kernel configuration will include an object file related to the board into the kernel. This will provide several board-specific initialisation functions, in particular `board_init()`, which is called quite early during boot by code generic to all 4xx chips.

[2]On machines with an IO MMU or similar these functions also establish the necessary mappings there.

Remaining problems and development to do

As usual, of course, there is still work to be done in fully supporting 4xx chips.

One of the biggest areas for work on 4xx support is in power management: at the moment Linux has only quite limited support for the power saving features of the 4xx chips. In particular the drivers for a number of the on-chip devices need updating to make use of those device's features for saving power.

A number of things in the 4xx support, particularly the handling of on-chip devices and initialisation for specific boards, are not implemented as cleanly as they could be. The situation is gradually improving though, which should make life easier when adding support for new 4xx chips and boards.

As discussed previously, the support for 4xx chips in the 2.5 kernel is quite incomplete at the moment. A large part of the reason being that people building particular embedded devices tend to work from 2.4 because they want to use a stable kernel. However, integrating 4xx support into the mainline 2.5 kernel will be important for making 4xx devices "full citizens" of the Linux world. One of the major tasks for 2.5 is to integrate the handling of 4xx on-chip peripherals into the new unified device model being implemented in 2.5.

There are some other aspects of the 4xx chips which are not supported and which it's unclear how to support, or even whether supporting them would be useful:

Currently Linux makes no use of the 405GP's on-chip memory (OCM)[3]. Since this memory is as fast as cache, it might be possible to improve performance by doing so. However, the OCM is only 4kB so not a lot can fit in there. What to put in there to see any benefit will obviously depend on what the machine is actually being used for - and it's quite likely that the most interesting code or data from this point of view will be in user space. Hence using the OCM really has to be implemented by the integrator of a particular embedded device. That said, it might be useful to provide a device driver which allows the OCM to be mapped into userspace.

Acknowledgements

Thanks must go to IBM, my employer, for their support of this paper. Thanks to Martin Schwenke and Paul Mackerras for proofreading.

Bibliography

[1] `linuxppc_2_4_devel` kernel tree, `bk://ppc@ppc.bkbits.net/linuxppc_2_4_devel`.

[2] IBM Corporation, "PowerPC 405GP Embedded Processor User's Manual", Seventh Preliminary Edition, 2000.

[3] Patrick Mochel, Linux Kernel Driver Model Documentation", `http://www.kernel.org/pub/linux/kernel/people/mochel/doc/`, 2001.

[3] Well, actually, it is used in a few instances for obtaining parameters from the firmware/loader at boot time, but there is no substantial use.

A Networked Loudspeaker

Jan Newmarch
School of Network Computing
Monash University
`jan.newmarch@infotech.monash.edu.au`

Home networking is arriving slowly. This paper reports on a project to stream audio around the house using wireless networking. The audio stream is received by a loudspeaker running embedded Linux with a power amplifier

Introduction

Two of my favourite pastimes are listening to music and cooking (with corresponding eating). These are not always compatible: like many people I have my hifi equipment in a room such as a loungeroom, well removed from the grease and heat of a kitchen. In my current flat the lounge and kitchen are separated by a long corridor, so that when I am working in the kitchen the sounds from the lounge are attenuated, to say the least.

A pair of speakers in the kitchen is the answer. One can buy "party speakers" which are hardened to withstand the outdoor environment, and so could probably cope with the kitchen environment. Most amplifiers have outputs for two or more speaker systems, so the simple solution is just to run an extra pair of speaker cables from one end of the house to the other. Cost: maybe ten dollars, depending on the quality of the cables.

In a previous house this is what I did, running the cables under the house. In my current flat there is no easy underfloor or intra-wall access, so the cables will have to run visibly. Untidy.

In addition, this relies on a device-centric view, based on the capabilities of the amplifier. The amplifier must have spare speaker outputs. If I change the amp, it has to support the same number of outputs. It is non-extensible, in that I can't just keep adding speakers to one room after another or placing speakers out in the garden to keep the gnomes happy.

My amplifier is also non-extensible in that it takes a small number of inputs: phono, cd, tape and tuner. I can plug the TV audio output into one of these, but that doesn't leave room for much else.

On the other hand, my PC is now a source of audio signals: with its soundcard I can use it to play CDs, and (if pressed) use the microphone input for karaoke while playing VCDs on the DVD player. More interestingly (and tunefully) it can be used to take streaming audio input from internet wireless stations, or to download and play MP3 files. With a range of bus types (SCSI, PCI, firewire, etc) it can support outputs and inputs to and from an increasing variety of devices and home appliances, with no limits to the number of connected devices.

PC as audio centre

There is a strong possibility that the centre of a home audio system will shift from the amplifier with a small number of inputs to the PC with a wider set of inputs. The PC with a soundcard can handle both input and output digital signals, store them and manipulate them in various ways.

The amplifier will not disappear, but may be split back into its roles of pre-amplifier and power-amplifier.

The pre-amplifier will be responsible for taking inputs from analogue sources such as phono, radio or microphone, applying the correct frequency response correction and outputting it in a form suitable for digitisation by a soundcard or other dsp. In time, the various sources will produce digital output directly, in the same way that the CD has largely taken over from the 12 inch disk. The pre-amplifier stage may eventually vanish from home audio systems, but currently it still plays a major role in producing a standard output signal from varying inputs.

The power amplifier will be needed to produce high-power signals to drive loudspeakers. Analogue amplifiers need to be fed with a suitable signal, such as that produced from a pre-amplifier (or soundcard) and analogue speakers need to be fed the output from analogue power amplifiers. These may give way to digital amplifiers driving digital loudspeakers eventually.

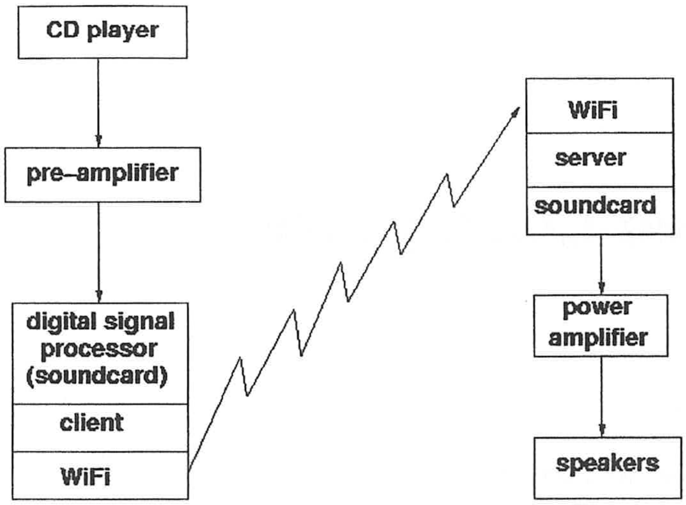


Figure 1: Architecture

Once you separate the amplifier into pre-amp and power-amp and introduce a digital stage in between, then the computer becomes a natural intermediate stage. It can take analogue signals through the soundcard, and also take digital signals directly from digital devices and from the internet.

There are also other benefits: PCs can be scheduled to run programs at arbitrary times. It is easy to set up an `at` or `cron` job to record an internet radio broadcast; much easier than programming most VCRs for a future recording.

A network loudspeaker

A computer can be used to accept and control audio input. Typically this is just used to send the audio back out through the soundcard to a set of PC speakers, which contain their own power amplifiers. Distributing audio in a home situation does not seem to have been addressed by vendors, although many of the ingredients are present in current systems

- Multimedia or powered speakers can be directly connected to soundcards, as mentioned above. The quality of sound produced by most PC speakers is fairly poor, even on "high-end" PC speakers costing several hundred dollars. There are only a handful of PC speakers of genuine hifi quality
- Firms such as Jaycar sell 2.4Ghz wireless transmitters and receivers designed to move TV and audio signals around the house. There is no amplification at receiver end, so to drive loudspeakers a power amp is still required. Also, it is not clear from the publicity material whether any attention has been paid to interference from other devices using this frequency (Bluetooth, WiFi, microwaves, etc)
- Wireless headphones are quite common, but they tend to suffer from directional effects, and anyway have no power amp stage

Once sound is digitised on the computer it can be sent on the network to other devices, and of course this is the basis of internet radio, IP telephones, streaming video, etc. In the home it is even easier to do this since the bandwidth of a home network (say on ethernet) is much higher than the internet. It is not even necessary to compress the sound: an audio CD delivers data at 1.35Mbps which can be sent on a 10Mbps ethernet without problems (four concurrent streams might cause problems, though).

Either wireless or wire connection can be used. IEEE 802.11b (WiFi) can broadcast at 11Mbps, about the same as 10Mbps ethernet and certainly enough for audio. Once received, the audio signal can be fed into another soundcard with output to a power amplifier and a pair of loudspeakers.

I have built a proof-of-concept wireless speaker that uses standard internet communication protocols, standard (free) software and standard hardware to receive an audio signal and play this out through a power amp built onto a loudspeaker.

The loudspeaker is built from low cost components:

- An old PC with 120Mhz processor and 400M disk was cannibalised by throwing out most unneeded hardware such as floppy disk, monitor, keyboard, etc. All that was kept was the motherboard, power supply, soundcard, disk and ethernet and wireless network cards. The disk can eventually be replaced by flash ROM, but in the meantime allows easier experimentation with software. Cost: $200 for the PCI wireless card

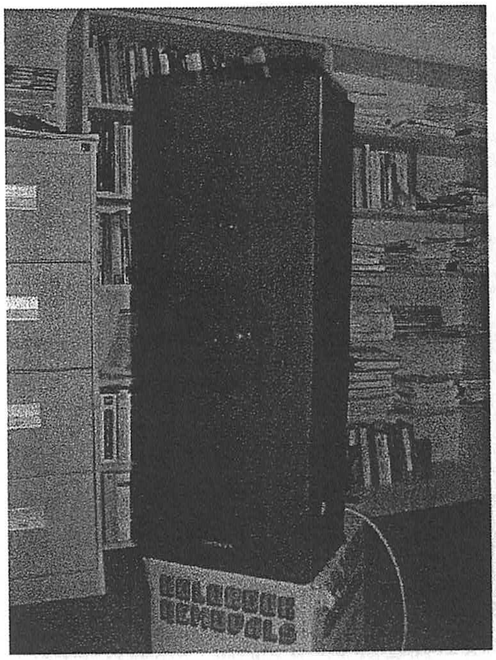


Figure 2: Front of speaker

- A power amplifier for a car audio system was bought secondhand for $40. These range new from $100 to thousands of dollars, depending on quality and power. These amps only require a 12 volt power supply, and the PC power supply has a 12V, 15A output
- A pair of secondhand speakers was used - cheap enough to be screwed together so that the back panel could be removed. More expensive speakers have sealed cabinets

An ordinary PC motherboard is fairly large (30.5cm by 21cm) since it needs room for memory, processor and several PCI and ISA slots. Smaller versions are available with fewer slots (Micro ATX form factor, 24.4cm by 22cm). A minimal motherboard for a speaker system could be quite small (for example, EMAC Inc sell an Embedded Linux Starter Kit with ethernet - but not wireless - on a 14cm by 10cm board). For now, I used a speaker large enough to fit the power amplifier inside, and with a back panel big enough to screw the motherboard, hard disk and power supply to it. The power supply for both the amplifier and PC is the original PC power supply which can deliver a 12 volt output for the car amplifier.

Open source software

The PC is a 120Mhz Pentium with 32M RAM. This runs a 2.4 Linux kernel (from Redhat 7.2). By building a fairly minimal custom installation less than 400M of disk space was required. This included a full C development environment, emacs and Perl (just in case). This could be reduced drastically - some embedded Linux's require just 8M of ROM. Because I use a hard disk and there is little possibility of soft shutdown (no keyboard, no mouse) the `ext3` journalling file system is used on the disk to avoid time wasting file system checks on boot.

For the proof of concept a simple TCP client/server pair of programs were written in about a page each of C. The server runs on the speaker, and a client on a sound source such as another Linux box will connect to it and stream audio. Each of the client and server use `sox` to control the sound cards, reading from and writing to them. The Sun `au` format was used since this doesn't need header information including the length of the stream. However, `sox` will support a number of other formats.

The sound drivers and network drivers (including the wireless driver) were all standard drivers with a Linux system. The hardest part was figuring out the keystrokes to run the BIOS `setup` program to tell the BIOS there was no floppy drive (for this old BIOS, it was the `del` key on the numeric keypad).

By use of open source software, the development time was about one hour to write the client and server. This can be used as the basis for other software versions, using e.g. multicast or various compression algorithms.

The project actually spanned several weeks: stripping down the old PC; trying to find motherboard mounting brackets and finally drilling them out of the old PC; trying to find a suitable pair of speakers, big enough to hold a motherboard and also with a detachable back panel; getting cheap wireless cards (the "computer swap meets" will

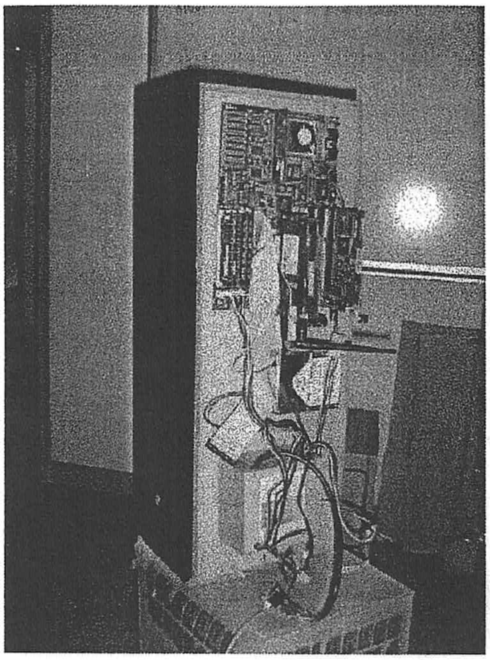


Figure 3: Back of speaker

sell old ethernet cards for $5, but none of the vendors are yet selling new wireless cards); finding Linux drivers for PCI wireless cards supporting Ad-Hoc mode (most are PCMCIA drivers for Managed mode); etc.

Results

Shielding is an issue: the disk drive generates enough of a magnetic field when accessing the disk to produce crackling on the speaker. Even using the keyboard on a nearby computer produces "clicks" on the speaker.

The sound quality is okay for a novice ear (like mine). However, if the gain on the amplifier is turned up and the signal is turned down then a "fluttering" effect can be heard on the speaker. The cause of this was tracked down to interence between network cards and the sound card, similar to the interference effects noted above. Other possibilities that were investigated and ruled out included:

- It occurs with either the power amp or a pair of headphones connected to the speaker (ruling out interference effects into the power amp)
- With ethernet card, it appears to be at about the same frequency as the packet light on the hub, so is related to packets traversing the network
- The initial version used TCP, which sends an acknowledgement of each TCP packet. However, a UDP version also displayed the same effect and UDP has no handshaking. The UDP version was okay when transferring sound from a CD player, but lost packets badly when reading from the sound card. However, one interesting effect was observed from the UDP version: the ethernet card transferred packets of 4kbyte blocks, the wireless in 8kbyte blocks, and the fluttering occurred at twice the frequency with the ethernet card as against the wireless card
- Next, the effect of context switching on each PC was investigated. On both machines a pipeline was used: one to read from soundcard using `sox` and send down a pipeline to a process writing to the network, and on the other end one process reads from the network and writes to a pipeline with `sox` at the other end. On each end there is context switching between the processes involved in the pipeline. `sox` was modified to include network code to write to the network as a client or to act as a server reading audio from the network. This replaces pipeline I/O with direct network I/O and avoids the context switch.
- The fluttering is independent of the volumes set on source and speaker sound cards. When another sink - a laptop - is used to play the audio stream, the effect disappears (to my ear). This suggests interference into the soundcard from the ethernet card. This is supported by nearby keyboard use producing clicks on the speaker. When a higher quality soundcard was used the fluttering was reduced, confirming that shielding on the soundcard is a significant issue

The uncompressed data transfer rate for 44.1Mbps sampling is about 1.5Mbps, as expected (this is content data rate, not including TCP, IP and ethernet packet data). This is achieved comfortably using twisted pair between ordinary ethernet cards. However with wireless cards it is a different matter, and probably reflects the maturity of implementations of wireless drivers

- With two D-Link PCI cards using the `linux-wlan-ng-0.1.13` driver (the only one that currently supports the D-Link PCI card in Ad-Hoc mode) the streaming dies after a couple of packets
- With a D-Link PCI card in the speaker and a Toshiba Wireless card using the `wvlan_cs` driver in a laptop, a data transfer rate of only 400kbps can be achieved. Good enough to stream MPEG compressed data, but not good enough to stream uncompressed CD quality data - packets just get lost, and the sound is *very* fragmented
- With an Aironet card in the speaker using the `airo` driver and the Toshiba card and `wvlan_cs` driver in the laptop there is no problem, and CD data can be streamed at 1.5Mbps without loss

The current poor results with the D-Link card are difficult to explain right now. A pair of D-Link cards can perform an ftp file transfer at 4Mbps, but can't cope with 1.5Mbps audio data. Perhaps it is related to packet size, which varies between 1-8kbits.

Further developments

The proof of concept needs to be replaced by a better prototype. This will hopefully be performed in conjunction with a loudspeaker and/or amplifier manufacturer. There are issues both from the hifi viewpoint and the embedded systems viewpoint

- Resolving the bandwidth issue for wireless cards
- Squeezing hardware components down to a small enough size to fit inside a loudspeaker cabinet
- Keeping good acoustics in a speaker cabinet filled with PC hardware, cards, power supply and power amplifier. Hopefully experience in powered speakers will help here
- Protecting the transport mechanism from interference from the loudspeaker: placing an 80 watt speaker coil and power amplifier next to a wireless card requires some shielding (of the power amp at least!)
- Devising a suitable flash ROM system to eliminate the need for a hard disk

There are various middleware initiatives such as HAVi and CEBus which provide a software model of home consumer electronics devices. These do not quite fit the network loudspeaker - neither have this concept, and neither middleware has addressed wireless networking. Integrating this into these middleware models could make the network speaker more useful. It could lead to software abstractions suitable for Jini or Web services.

The five or six speaker home theatre systems use a larger variety of control mechanisms than for simple stereo speakers:

- Size of speakers
- Placement of speakers
- Relative volumes

There do not as yet seem to be suitable software models for theatre speaker systems.

The network speaker provides a cheap and useful device for experimenting with streaming technologies.

References

http://jan.netcomp.monash.edu.au/

What is that CONFIG_HAMRADIO thing anyway? – A Linux users guide to Ham Radio

Hugh Blemings
hugh@blemings.org
VK1YYZ

Introduction

Ham Radio is about hobbyist experiments with radio. This experimentation takes many forms, seeing how far a signal can travel, building antennas, building radios, trying out new modulation modes and having some fun along the way.

In this paper the terms Ham Radio and Amateur Radio are used interchangably. One could argue that "Ham" is more common in North America. What follows is just one perspecitve on the hobby and the things that make it interesting.

History

Ham Radio can trace its roots to the earliest days of radio experimentation. Marconi, Hertz et. al. were the earliest experimenters and inventors in the field and could (perhaps at something of a stretch) be considered the first "hams". Amateur radio was first recognised by government agencies in the early 1900s when it became possible to apply for an experimental radio license. This in time turned into a specific Amateur Radio license for non-professional, non-broadcast oriented operation. More rigorous treatments on the history of the hobby abound on the web.[1]

The Cooks Tour

Like any hobby, Ham Radio has it's own jargon, terminology and quirky bits. To assist the reader in making sense of later sections a tour of same is provided.

Callsigns

Amateur callsigns follow an international convention whereby the prefix is based on letters/numbers assigned to the country, the remainder of the callsign by local regulation. For example, Australia is assigned the prefix group VA-VZ (that's why our aircraft have callsigns VH-XYZ) - Amateur licenses are allocated VK. The third digit is a number corresponding to the state of operation (0 - Antarctica, 1 - ACT, 2 - NSW, 3 - VIC, 4 - QLD, 5 - SA, 6 - WA, 7 - TAS, 8 - NT, 9 - Islands). The remaining two or three characters are letters according to the class of license. My callsign (VK1YYZ) identifies me as being in the ACT and holding a "Limited" license (VHF bands only)

Frequencies and Bands

Hams have particular ranges of frequencies (or "bands") allocated for their use. For the most part the bands are common worldwide (or have reasonable amounts of overlap) and agreed upon by international treaty. Hams will tend to refer to bands by their wavelength such as 70cm, 80m, 40m etc. frequencies being referred to only when the

[1] This site at the ARRL web page is just one example `http://www.arrl.org/tis/info/history.html`

Frequency	Band	Notes
1800-1875 kHz	160m (MF)	Large antennae sizes tends to limit number of active operators
3.5-3.8 MHz	80m (HF)	Popular but somewhat noisy long distance band
7.0-7.3 MHz	40m (HF)	Popular long distance band
14.0-14.35 MHz	20m (HF)	Arguably the most popular HF band
18.068-18.168 MHz	16m (HF)	Underutilised band but great for long distance work
28.0-29.7 MHz	10m (HF)	Last of the "HF" bands, popular for SSB and FM work
50.0-54.0 MHz	6m (VHF)	The "magic band" - good long distance performance at times
144.0-148.0 MHz	2m (VHF)	Popular band for local work, satellite, packet, APRS etc.
420.0-450.0 MHz	70cm (UHF)	Popular for local work as well as satellite, EME etc.
1240-1300 MHz	23cm (UHF)	Relatively little use but popular for higher bandwidth modes
2300-2450 MHz	12cm (UHF)	Shared w/ISM - same frequency range as 802.11 wireless
10.0-10.5 GHz	3cm (SHF)	Top end shared with ISM band
248.0-250.0 GHz	1mm (SHF)	Getting close to light...

Table 1: Sample of Bands allocated to Amateur Service in Australia

specifics are important to the discussion. Some bands or sections of bands allocated to the amateur service are on a secondary basis, that is to say that hams must accept any interference from other users of the band.

Table 1 has a listing of some of the bands available to Australian amateurs.[2] There are half a dozen or so allocations in the SHF bands not shown - experimentation is done on these frequencies but it tends to be pretty specialist in nature due to the difficulties in fabricating circuitry for such short wavelengths.

Modulation methods

Modulation is the process of impressing information on a radio signal or carrier. A vast range of modulation techniques exist, most of which will have been tried by ham operators at some point. The more common modes include

- Continuous Wave (CW) - pretty much the original mode of communication and the "proper" name for Morse Code transmission. The carrier is modulated in an on-off fashion by the morse key or an electronic equivalent thereof. For reception by ear still the best mode in weak signal conditions, only bettered by some recent digital modes that rely on computer/DSP techniques, very low baud rates, forward error correction and esoteric but frightfully clever modulation schemes.

- Single Side Band (SSB) - most common for medium and long range communications. Characteristics of the transmitted signal make it particularly suitable for weak signal work. SSB is also the mode most commonly associated with ham radio by non-hams, it's the mode that has the whistling (hetrodynes) and squeaky voices. Used for both voice and data/digital modes.

- Frequency Modulation (FM) - more common for short range communication or working through repeaters. Most prevalent on VHF frequencies (50 MHz and up). In good signal conditions voice quality is similar to telephone but degrades quite markedly if signal conditions are poor. Like SSB, used for voice and digital modes.

- Amplitude Modulation (AM) - less common nowadays but a favourite on the low HF bands (3.5MHz and below)

- Spread Spectrum modulation in it's various forms has long been an area of experimentation on the UHF and SHF/Microwave bands. Undergoing something of a renaissance due to the interest in digital/software defined radio.

Operating Modes

Hams speak of particular operating modes which can mean the modulation mode in use or what they're doing in a more general sense.

[2]Source - Wireless Institute of Australia (WIA) Callbook 2002

DX

DX or long distance operation is one of the long standing modes of operation for hams. Frequencies of operation range from the low HF bands up to microwave. Within the spectrum of DX operation people may use voice, CW (Morse) or data modes. Stations may use high power (up to 1.5kW or more) or very low power (so called "QRP" operation) - below 5W.

At HF frequencies long range communication relies on radio waves being refracted by layers in the ionosphere allowing the signals to "skip" over great distances. Given the right conditions an operator in Australia can talk to fellow hams literally anywhere in the world. Antennae for HF work can range from simple random wire strung up between two trees through to large directional ("beam") antennas which may have multiple elements up to 20m long each.

At VHF frequencies the ionosphere doesn't tend to refract radio waves to the same degree so DX operation at these frequencies is usually reliant on other modes of signal propagation. Meteor scatter makes use of fortuitous changes to the ionosphere from meteor showers. Aircraft enhancement relies on brief periods when commercial aircraft are between two stations to bounce signals off the aircraft itself or the resulting temperature inversion from the exhaust gas.[3] Earth Moon Earth (EME) uses a mixture of high power, high gain antennae and sensitive receivers to bounce signals off the moon. Satellite operation makes use of man made amateur satellites which have transponders in them similar to commercial satellites in either low earth or geosynchronous orbits.

Voice

Voice is still the most common for ham operators. At one extreme you have people working DX using SSB through to "Rag Chewing" - talking about the weather and other fascinating subjects on FM through the local repeater on the way home from work.

As a weak signal mode voice does reasonably well, the human ear being well tuned to discerning speech in situations where there is a great deal of ambient noise.

CW

CW or Morse Code is the oldest mode of radio communication. It is still popular today both among die hard operators who wouldn't use anything else and newcomers who enjoy the challenge, romance or excellent weak signal performance CW affords.

Morse can be sent in many ways, a simple morse key, a paddle (or bug) which in conjunction with a "keyer" generates the dits and dahs automatically or via computer. Morse is usually received by ear though software and hardware/firmware based decoders are also widely used.

Digital/Data Modes

Digital or data based modes of operation have had their place in Ham radio from well before computers were commonplace and certainly before the Internet. The earliest data mode could be argued to be CW as it is binary in nature but most would consider Radio Teletype (RTTY) to be the first true digital mode. Since anything digital is likely to interest your average Linux hacker Digital/Data modes will be covered in more detail in the next section.

Mobile/Portable/Fixed operation

While not really different modes per se there is variety in how and where people set up their stations. Home or fixed operation is the most common, a corner of a bedroom or spare room/shed forming the ham's "shack" (originally from the maritime use of "radio shack" to mean the radio room on a vessel)

Mobile operation is common, particularly on VHF bands and up using FM voice and repeaters. HF mobile isn't as common but there are folk around who even use CW while mobile (though presumably they're stationary at the time...).

Portable operation is popular - this may be as simple as using a small handheld transceiver or as complex as setting up a comprehensive DX station on top of a hill for a weekend contest.

Contests, Records, Honour Rolls & QSLing

There are many organised contests that run during the course of the year. The general concept is to make as many contacts as possible in an allotted time frame - usually these contacts are very brief consisting of a signal report and a serial number. There are CW only contests, mixed modes, HF only, VHF/UHF only etc. etc. Some contests award more points for contacts made from portable or mobile stations, others make no such distinction.

---

[3] I'm reliably informed that the actual cause of enhancement is a subject of ongoing debate...

Alongside contests there are various records that are recognised on a local or worldwide basis for such things as the longest distance contact on a particular frequency and mode. For anyone who has dabbled with extending the range of 802.11 wireless links the UHF/microwave records are particularly interesting - the current record for the 2.4GHz band is 3,980km for terrestrial communications and 16,480km for EME.[4] To be fair this is for CW which has rather lower bandwidth and rather higher noise immunity than wireless ethernet...

Honour Rolls are a variation on contests/records - usually they are based on making contact with a certain number of different countries or callsign areas. One of the most well known, the DXCC (DX Century Club) is awarded for contacting more than 100 different countries/call areas. There are variations on the DXCC for contacts all being made using one band or mode of operation, CW or voice only etc, etc.

QSL cards are postcard sized cards that are optionally exchanged between hams to confirm a contact. Information about the frequency, mode, date, time of the contact are included along with (traditionally) details of the station, location etc. Electronic QSLs (such as confirming a contact by email) are becoming more common though these are not yet widely accepted for contests.

Licensing

Like most uses of the radio spectrum, amateur radio operators are required to obtain a license. There are different grades of license which dictate the frequencies/bands you are able to use as well as the operating modes. To obtain a license it is necessary to sit various exams that cover Regulations, one of two grades of theory and in some cases Morse Code reception though it is expected the latter will be withdrawn in the next few years.[5] The entry level ("Novice") theory exam should be within the capabilities of most people that have a background or hobby interest in electronics with a little revision on radio specific areas.

One of the few things you can't do with a ham license is use it for direct commercial gain, this is one of the basic tenets that distinguishes the hobby from commercial and broadcasting licenses.

Data/Digital Modes

Voice, CW and other analogue modes are still very popular. However, for the average Linux user digital or data modes are likely to hold the most interest. As it is based on on-off keying, CW can be argued to be a digital mode however I follow the more conventional view that it's analogue as it's usually received by ear.

Radio Teletype (RTTY)

Teletype is one of the oldest digital modes of communications, dating back to the early 1900s. Radio Teletype[6] seems to have originated around the 1920's though it did not become popular in Ham circles until the fifties or so. Early Ham Radio RTTY experiments made use of mechanical teletypes and were operated at 45 baud on HF frequencies using Frequency Shift Keying (FSK).

RTTY is still in use today both in ham and commercial circles. It is one of the easier modulations to decode and there various Open Source packages that will do so - these will be covered in more detail in the next section.

Packet Radio (aka AX25)

Amateur Packet radio first saw the light of day in Montreal, Canada in 1978.[7] The AX25 (Amateur X-25) protocol[8] was quickly accepted as the standard protocol. The Vancouver Amateur Digital Communication Group (VADCG) developed a Terminal Node Controller (TNC) in 1980 which was built upon by the Tuscon Amateur Packet Radio Group[9] and others to arrive at the so-called TNC-2 standard used today. The TNC was designed to sit between a radio (usually an FM transceiver operating in the 2m band) and a dumb terminal or computer. These early TNCs were based around the Z-80 CPU and included a 1200 baud modem (7910 chips then TCM-3105 devices) and the required software stack for AX-25 protocol and various application layer functions. The availability of TNCs caused quite a flurry of activity despite the low (by today's standards) transfer rates. Bear in mind this predates widespread access to the Internet, the WWW and pervasive email by nearly ten years.

Hams rapidly set up packet radio networks that incorporated hierarchical addressing schemes so that packet messages could be sent over large geographical areas. In addition to real-time keyboard to keyboard communication,

[4]Source G3PHO's site at `http://www.g3pho.free-online.co.uk/microwaves/records.htm`

[5]Andrew Davis' Australian Amateur Radio FAQ `http://members.ozemail.com.au/~andrewd/hamradio/hamfaq.html` is an excellent source of information on licensing and most other facets of Amateur Radio.

[6]`http://www.rtty.com`

[7]`http://www.tapr.org/tapr/html/Fpktfaq.html`

[8]AX-25 is pretty much X-25 with larger address fields to allow a standard amateur callsign represented in ASCII to be used with room for a four bit Sub Station ID (hence VK1YYZ-1, VK1YYZ-15 etc.)

[9]`http://www.tapr.org`

most TNCs incorporated a rudimentary mailbox system such that you could connect to a friend's TNC and leave a short message. In much the same way as we check for email when returning home today, in the mid-80s you'd stick your head into the shack to see if the yellow message LED was blinking. Cool stuff.

Bulletin boards became quite popular too, these were usually DOS based in the early days with Jean-Paul Roubelat's famous F6FBB[10] package being by far the most popular.

Digipeaters became quite common allowing a station to connect in "realtime" much further afield. On a good day you could connect from Melbourne to Sydney entirely over the packet network. RTTs were pretty long though...

1200 baud is still the most common data rate today as it's easily accommodated on a standard FM channel and transceiver audio path. Standards exist for 4800, 9600 and higher speeds with a commensurate increase in the channel bandwidth required. Higher baud rates usually require modification to the transceiver itself to avoid the signal shaping done for voice by some radios.

TCP/IP has also been used with AX-25 as the transport layer. Phil Karn's KA9Q NOS package was in widespread use and was also one of the first implementations of TCP-IP on small systems.[11] Quite large packet based TCP/IP networks were built using the 44.0.0.0 class A network assigned to the Amateur Radio service worldwide - no packets were routed on/off the Internet to the 44 network however. There is also an ampr.org domain for ham use.

APRS

APRS(Automatic Position Reporting System) was developed by Bob Bruninga, WB4APR.[12] It makes use of the AX-25 broadcast frame to encapsulate position and other tactical/realtime data in a single packet which is transmitted as a broadcast packet. What this amounts to is that by combining a GPS receiver, 2m FM transceiver and an APRS encoder you can broadcast realtime status information for tracking vehicles and the like. This can be rather for fun [13] as well as being very useful when coordinating vehicles in civil emergency situations (this being one of the catalysts for APRS development).

A worldwide network has been set up that makes use of the Internet as a backbone and allows realtime APRS data to be gated from local radio channels to central servers that can be queried or connected to over TCP/IP[14]

As an aside, Bob Bruninga also oversaw the development of PC-Sat - a low cost Amateur Satellite that digipeats APRS data and by nature of it's low earth orbit can be accessed from quite modest equipment (5W handheld radio and small vertical antenna for example). It's antennae were constructed out of measuring tape blades.

PSK31

PSK31 is an increasingly popular mode for HF work. It stands for "Phase Shift Keying, 31 baud" but this really only tells part of the story.[15] Suffice to say it's proving very popular for long distance digital communications. You're not going to use it to pull down the latest Debian ISOs but for keyboard to keyboard work it takes some beating!

Amateur Satellites

The amateur fraternity have a long association with space. The first amateur satellite, OSCAR-1,[16] was launched in 1961 as a piggyback to a USAF satellite launch. This satellite like all amateur satellites was designed and built by hams and funded by donation. OSCAR-1 was a very simple device weighing some 4.5 kg and having a single antenna and battery powered transmitter that sent "HI-HI" in morse code on the 2m band. The speed at which the morse was transmitted varied according to the temperature of the SV forming a crude telemetry system. The transmitter output was a mere 140mW which drained the (non-rechargable) batteries in around 3 weeks. By that time more than 570 amateurs in 28 countries had reported hearing OSCAR-1.

Modern "AMSATS" are rather more sophisticated and include such things as:

- Computer control of major SV parameters which can be accessed by ground based controllers.
- Attitude and orbit control systems.
- Packet or data capable transponders that can operate in a store and forward or digipeater capacity.
- Earth pointing cameras.
- GPS receiver systems for space borne GPS experiments.

---

[10] `http://www.f6fbb.org/`

[11] `http://people.qualcomm.com/karn/code/ka9qnos/`

[12] `http://web.usna.navy.mil/~bruninga/aprs.html`

[13] See where my car is - `http://www.findu.com/cgi-bin/find.cgi?VK1YYZ-7`

[14] try 'telnet aprs.net.au 10151' and/or software like Xastir - `http://www.xastir.org/`

[15] For the complete story... `http://www.psk31.com`

[16] For Orbiting Satellites Carrying Amateur Radio

- Packet BBS.
- Smart transponders that can limit the amount of downlink power allocated to strong signals. Most AMSATs have several combinations of uplink/downlink frequencies.
- Sensitive receivers and relatively high power transmitters permit operation with some AMSATS with quite modest stations.

The International Space Station has a permanent APRS digipeater system as well as a modest ham shack so that astronauts that have their ham license can operate from space. It is also quite common for hams to be operational on Shuttle missions. Both these activities are coordinated by SAREX - Space Amateur Radio Experiment.[17]

A couple of pertinent URLs - AMSAT-VK `http://www.physics.usyd.edu.au/~ptitze/amsatvk/` and AMSAT-NA `http://www.amsat.org/`

IRLP

Internet Radio Linking Protocol[18] is a system developed by David Cameron, VE7LTD that allows voice band ham repeaters to be linked via the Internet. In Australia most repeaters linked by IRLP are in the 70cm FM band which has added a whole new facet to the morning drive in to work. It is not uncommon to hear local hams chatting with other stations on the other side of the world. For hams that are unable to operate on the HF bands this is quite a boon - I've had chats with people in Ottawa, Chicago and San Jose on the way in to work of a morning.

Software Radio/Software Defined Radio

Joseph Mitola III coined the term "Software Radio" in 1991 and defines it thus;

> A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband DAC and then possibly upconverted from IF to RF. The receiver, similarly, employs a wideband Analog to Digital Converter (ADC) that captures all of the channels of the software radio node. The receiver then extracts, downconverts and demodulates the channel waveform using software on a general purpose processor.[19]

What this means in practice is moving the digitisation process (or digital to analogue in the case of a transmitter) as close to the antenna as possible then doing all the demodulation and other clever things in software. In some respects this is an extension of what has happened with conventional telephone modems which moved from being a predominantly analogue device to more or less completely DSP based around the time rates greater than 2400 baud became commonplace.

The availability of cheap, high (computationally) powered processors has made software defined radio practical for even the home experimenter.

Others

There are many other modes of Digital/Data operation, too many to cover here in this overview. Google is your friend if you want to learn more.

Equipment

A basic ham station may consist of nothing more than an antenna, radio and associated cables. Like most hobbies there is a myriad of options available to fit different areas of interest, budget and the amount of room in your back yard. New versus secondhand, purpose built commercial gear versus "homebrew", basic versus sophisticated. There are many sites on the web that discuss all of these facets and more but a quick overview is in order.

---

[17] `http://sarex.gsfc.nasa.gov/`
[18] `http://www.irlp.net`
[19] `http://ourworld.compuserve.com/homepages/jmitola/whatisas.htm`

New versus Secondhand

A good market in secondhand equipment exists and for the most part a 10 year old radio will perform just as well as it's contemporary equivalent just with less flashing lights or sophisticated features. It's worth enlisting the assistance of an experienced ham if considering secondhand gear - there are some dogs out there as well. Your local ham radio club is a good starting point here,[20] they may even run local classifieds or buy/sell nights. If you're based in Australia, the website run by VK2CA[21] is a great starting point for classifieds, in my experience.

If you're at the dipping your toe in the water stage secondhand can be a good way to go even if you just want to listen to begin with. It is legal to own transmitting equipment if you don't have a license provided you only use it to receive. Penalties are quite severe (AU$10,000 fines and more) for illegal transmission - in short, don't.

Surplus commercial equipment (e.g. ex-taxi two-way radios) can be a good buy too as they can often be re-programmed or re-tuned to suit ham bands. Newcomers will definitely need guidance here but don't be put off by the fact that these sorts of radios look quite utilitarian - their performance is often very very good indeed.

Radios

Your choice of radio will be dictated by your area(s) of interest and class of licence you have (or intend obtaining). If your interest is mostly in talking to local hams on VHF or data modes a modest handheld (walkie-talkie) style unit may be a reasonable starting point, these can be obtained secondhand for AU$200 and up, new starting at around AU$350.

Units designed for mobile operation can be pressed into service in a base station situation as well - you'll need to provide a suitable DC powersupply (13.8V @ 5A or more) again these can be purchased new or secondhand for HF, VHF and UHF bands.

There are many different options for home or base station radios as well. Some contemporary transceivers cover all the bands from 160m to 70cm and have so called multi-mode capability (SSB, FM, CW, data, etc.) Older base station rigs are still quite usable though you need to be wary of units that have valve finals as these can be somewhat temperamental for the uninitiated.

Many people still homebrew their own equipment, this can range from basic low power CW (Morse Code) transceivers through to exotic software defined radios.[22] If you're interested in the electronics side of things and are handy with a soldering iron this is a great way to go. Not so good for folk desiring instant gratification though...

Antennae

Like the choice of radio, your choice of antenna will depend on areas of interest but also the environment you want to operate from. Like choosing a radio soliciting advice from other hams in your area is well worthwhile.

For mobile operation the most common antenna are vertical designs that differ in the method of mounting and operating bands. Assuming you're wanting to operate only on the VHF/UHF bands a simple quarter wave vertical mounted in the middle of the roof is very effective. Magnetic mounts and other no-hole options are available if you don't want to drill (which is most often the case!). I've had a very good run with a through glass antenna similar to those used for mobile phones.[23]

For base station or home use your options are many and largely dictated by what your circumstances allow. If you're on a residential block long wire antennas are usually an option, depending on your neighbours and council planners 30ft towers may also be OK... If you're in an apartment your options are reduced somewhat but there are many people who have worked the world from small antennae on their balcony or strung up on a window. Once again some research on the web will turn up many options.

Feedline

Feedline is the term used for the cable between an antenna and a radio. In most cases this is a coaxial cable similar to that used for TV antennae and 10-Base-2 ethernet. Open wire or ladder feed is also used under certain circumstances but these are somewhat specialised.

You will need some amount of cable between your rig and antenna. If the run is short say two or three metres then the choice of cable doesn't become critical until you start getting into the higher frequencies (say 430MHz and up). If your run is more than this you need to consider the cable type a little more carefully.

[20] The Wireless Institute of Australia website maintains a list of clubs organised by state here `http://www.wia.org.au/aushams/`

[21] `http://www.vkham.com/`

[22] The "DSP-10" 2m transceiver described by Bob Larkin, W7PUA is just one example. `http://www.proaxis.com/~boblark/dsp10.htm`

[23] It's a dual band design (2m and 70cm) made by MFJ Enterprises. A re-mount kit is available so you can move it to a new vehicle. Part numbers MFJ-1734 and MFJ-94 respectively. Bit of black heatshrink around the loading coil reduces whistle.

The basic problem is signal loss, particularly at VHF frequencies and above. Coaxial cables are lossy such that the humble RG-58 coax as used for 10-Base-2 ethernet is losing around 3dB for a 10 metre run at 430MHz or nearly 7.5dB at 2.4GHz (important to note if you're doing wireless LAN work). If you're trying to receive signals that are even a little bit on the margin this sort of loss reduces your ability to resolve the signal correctly - a nuisance for data modes in particular. It is fair to say that a loss margin like this is sufficient to make the difference between getting a contact and not.

At HF things are a little more benign so for 10 metre runs at (say) 14MHz you're not losing much. That said RG-58 still isn't held in particularly high regard for HF due to it's (comparative) fragility and low power handling capability.

Some rules of thumb then are:

- For temporary installations or short runs up to 2m/144MHz, RG-58 is OK, RG-8 (higher diameter) is better.
- RG-8 should be considered a minimum for permanent or high power HF work and 2m/70cm work of any kind where cable runs exceed a few metres.
- Permanent installations or low signal strength setups for VHF/UHF should consider using more exotic cables such as LMR-400 for runs over a few metres.
- High UHF and Microwave operation LMR-400, LDF-550 etc. are a minimum no matter what the length.
- Use a cable loss calculator such as the online one at Times Microwave System's website `http://www.timesmicrowave.com/cgi-bin/calculate` to estimate likely loss figures

The Linux Connection

Ham radio has had a long association with Linux. A number of well known Linux kernel hackers and contributors to Open Source Software are also hams.[24]

Linux Kernel Support

Initial ham radio support for the AX25 protocol appeared in the early 1.0.x series Linux kernels. Hams had long been using computers to act as routers, gateways, BBS/mail systems and the like as part of packet radio stations. Most of this was DOS based. With the advent of Linux an affordable, open source multitasking operating system became available. Hams quickly seized on the opportunity to press Linux into service in these sorts of roles.

As a result, the Linux kernel has native support for AX25, NetROM, ROSE networking protocols as well as many Ham-specific device drivers. The support is very nicely integrated to the point where most networking utilities have support for Ham-specific functions. For example `ifconfig` can be used to configure the callsign of an AX-25 interface as well performing more familiar TCP/IP configuration.

Open Source Ham Software

There is a wide range of ham radio oriented Open Source Software available.[25] The following list is by no means exhaustive but gives an idea of what is out there.

APRS

- Xastir is a nice APRS tracking/base station package for X windows. It can connect to APRS servers or to local AX-25 interfaces and TNCs. You can import maps, download track information etc, etc.
- aprsd is a daemon to handle APRS traffic. It can handle both local (radio) ports as well as TCP/IP connections and does all the stuff required to gate between them, connect to upstream AX25 servers etc. Hamish Moffat, VK3SB a ham based in Victoria contributed quite a bit of code to this daemon.

[24] Among the better known - Alan Cox (GW4PTS), Bdale Garbee (KB0G), Terry Dawson (VK2KTJ), Thomas Sailer (HB9JNX), Matthias Welwarsky (DG2FEF).

[25] One of the more comprehensive sites for Linux Ham Software is `http://radio.linux.org.au/` I've not provided specific URLs for the packages mentioned as they're either listed here and/or Google brings them up as the first or second hit.

soundmodem

Thomas Sailer's soundmodem package allows you to use a standard Linux compatible sound card to send and receive packet radio signals using a variety of modulation standards. This is a nice way to tinker with packet with minimal outlay and can be used for receive only if you don't yet have a ham license. A simple interface circuit to isolate your computer from the radio is all that is required.[26]

gnuradio

gnuradio is an open source package that provides tools for constructing software defined radios as discussed above. Out of the box it provides support for FM demodulation and spectrum displays and has an API defined to allow user defined modules to be readily added. gnuradio is in it's early days to some degree but people are already using it for real world applications.

g-psk31/kpsk

g-psk31 and it's successor kpsk provide transmit/receive functionality for the PSK31 mode under Linux. g-psk31 is GTK+ based. kpsk is based on Qt and seems to be the focus of ongoing development. kpsk amongst other things has a nice waterfall display and seems to be easy to get going. Like soundmodem it is soundcard based so you need only minimal additional hardware to interface with your radio.

predict

predict is a text based application for tracking satellites. It takes its input in the form of a text file containing Keplerian elements (keps) and allows satellite passes to be predict at arbitrary points in the future.

hamlib

Most contemporary commercial ham rigs have some form of serial port or remote control functionality. This ranges from basic programming of memory functions through to full remote panel operation. A group of open source developers have been working on a uniform library to provide control capabilities for a wide range of rigs.[27]

Do try this at home

Ham radio in Australia remains popular though the number of active operators is declining slowly, largely due to an aging ham population. Many new hams are licensed each year and the forgoing should show that there is a great deal of activity in leading edge areas.

Getting set up with a basic home station is quite straightforward and quite a lot of fun can be had with even a modest station and some tenacity.

For those with a technical/hacking bent ham radio can be a fascinating and rewarding hobby. I've been involved with it on and off now for nearly 20 years and still get a kick out of talking to another country, recovering a signal out of the noise or just yacking with friends on the local repeater.

Give it a go!

Close

Thanks to the Big Cool Guy (aka Martin Schwenke) for proof-reading, Rusty and sfr for suggestions about efficient ways of digging around in the kernel source for the early days of AX25 support, Chris Davis (VK1DO) for encouraging me to get involved in the first place and answering innumerable radio related questions over the years as a consequence. Finally but by no means least, thanks to Lucy and Rachael who make time not spent in front of radios so worthwhile.

Errata for this paper will appear at `http://misc.nu/hugh/ham4linux/errata.html`

[26] Ernie Mills, WM2U's site is nice and comprehensive for homebrew rig interface solutions `http://www.qsl.net/wm2u/interface.html`

[27] Some URLs for hamlib seem to be out of date relative to this one -`http://sourceforge.net/projects/hamlib`

KAME and IPv6 deployment

Jun-ichiro itojun Hagino <itojun@iijlab.net>
Research Laboratory, Internet Initiative Japan Inc.
`http://www.kame.net/`

Abstract

The KAME project was started in 1998 to provide BSD-licensed reference implementation for IPv6, IPsec and other recent Internet technologies. This paper tries to describe our motivation, where are we right now, and our future plans.

All of our source code is, and will be distributed under a BSD-like license.

Motivation, and the history of IPv6 and KAME

At IETF (Internet Engineering Task Force), the IPng effort was started in 1992 to re-architect IPv4, and make it possible for the Internet to grow much further [1].

IPv4 was designed in 1970s, and IPv4 address has only 32bits. 32bits may be sufficient for designs in 1970s, but not for 1990s and beyond. The theoretical limit of IPv4 address space is 4.2 billion, which is much less than human population on the planet. Actual upper limit is much smaller, due to the fragmented allocation of IPv4 address space - tens or hundreds of millions.

In 1995, the final candidate for IPng was selected, which is now called IPv6 [2, 3].

We, WIDE project (a research consortium for Internet technology, consists of 100s of universities and companies), thought IPv6 is the way to go, and implementation efforts were started. By 1996, within WIDE project we had 7 independent implementations of IPv6.

The independent implementations were good as a start (perform interoperability tests, identify common implementation errors, and identify protocol flaws), however, as time goes by we faced two problems. The first one is a human resource problem – each of the implementation had only one or two dedicated coders, and they do not have enough time to maintain their codebases. The second one is deployment problem – we felt an urgent need to integrate IPv6 stack into various BSD distributions, and to redistribute it under BSD license.

In IPv4 days, the BSD IPv4 stack was considered as a reference implementation. Interoperability tests were made against BSD IPv4 stack, people studied IPv4 from the code (like Stevens' "TCP/IP illustrated" books), BSD IPv4 stacks were deployed to confirm the scalability/dependability, and it improved the IETF standards. On the contrary, IPv6 was started with no (or very few) testbed codebase, there was no reference implementation, and we had almost no experience in deploying IPv6 codebases to wide-area network (not the tiny laboratory network).

Therefore, in fall 1997, we decided to start an effort to (1) integrate the codebases we have into one, to make it a single effort, (2) redistribute it under BSD license, (3) operate the code nationwide to confirm the scalability/dependability, and (4) integrate our codebase into *BSD operating systems to redistribute it widely and to make it a reference implementation. This is what the KAME project is about. The KAME project itself is focused into the implementation (coding) part of the above goals. Network operation part is handled by IPv6 working group in WIDE project.

What is "KAME"? What does it mean?

KAME is the Japanese word for turtle/tortoise. While we were working on our 7 independent implementations, one of the guy had a cute stuffed sea turtle. He (or she?) helped us debug the code by helping us calm down during sleepless debugging sessions and endless discussions. One of us (actually, me) suggested the integration project to be named after the stuffed sea turtle.

When the project was to become official, we considered changing the name of the project, as the name sounded too weird for foreign people. But we ended up keeping the name, as we could not come up with any names that can beat "KAME".

How is the KAME project funded?

KAME project is a child project of the WIDE project. WIDE project is funded by participating universities/companies, and partially by government (we get some government research grant). KAME project takes the similar route - KAME project has a small number of (7 at this moment) participating companies and universities. Salary for the project members (developers) were paid by the participating companies. We get some research grant from the government so that we can pay rents for our offices and such.

Status of the project

KAME IPv6/IPsec stack is integrated into all 4 *BSD projects (NetBSD, OpenBSD, FreeBSD and BSD/OS), as well as various vendor router products (like Juniper, Extreme, Hitachi and Fujitsu), commercial operating systems (Apple MacOS X) and embedded products (like Windriver's VxWorks). I think it is safe to say that we are successful in making a reference implementation of IPv6.

The winding road before the integration

In 1998, we focused in integrating our 7 independent implementations into one. We take the best part from each of the implementations, so we can say that the end result is pretty good. In 1999, we started contacting *BSD projects if they are interested in integrating our stack. At that time, there were other IPv6 stacks for BSD - INRIA and NRL, and *BSD project had hard time how to decide which one to take. Therefore, INRIA/NRL/KAME made an effort to integrate three stacks into one (another integration effort!). During the integration process, INRIA and NRL disbanded the projects due to funding and human resource shortage. Many of the good ideas and changes made by INRIA/NRL were integrated into KAME, therefore, we can honestly say that KAME is influenced (in a good way) by INRIA and NRL implementations.

The integration policy

Though IPv6/IPsec basic specs are set in stone, there are hundreds of internet drafts coming up for IPv6 and IPsec. There is a questions on how we should integrate these portions into *BSD.

We basically have two separate tracks - *BSD track, and KAME track. *BSD track has protocol supports for specs made it to RFC, and are needed for wide availability. KAME track implements various internet drafts so that we can identify protocol flaws. Some of the protocol drafts are really experimental, and need to be torture-tested.

For instance, Mobile IPv6 is yet to become an RFC, and every time a new internet draft is issued, header format and other things are changed. Therefore, it is not integrated into *BSD releases. Mobile IPv6 code is available in KAME distribution, which is distributed as a patch to the latest *BSD releases.

IPv6 deployment status in Japan, and worldwide

As many of you may be aware of, IPv6 deployment is now ongoing. In Japan, there are at least 5 ISPs which provides commercial-quality IPv6 connectivity services, and at least 20 ISPs which provides experimental-quality services. Most of the router vendors are shipping IPv6 into their products, or are running beta-test programs with selected customers.

There are a couple of roadblocks against IPv6 deployment at this point:

- More ISPs needs to become IPv6 ready, and start providing services, otherwise the customers cannot use it. In my opinion, ISPs needs to act proactively in deploying IPv6 network (rather than waiting for customer demands), since they need to gather operational experiences much earlier than customers, and they need to have a working backbone by the time customers start asking for IPv6.
- xDSL/L2 wholesale providers need to be ready for IPv6, as many of L3 ISPs depend on their services.
- Cheap/small routers for SO-HO/household needs to become IPv6 ready, however, price competition is extreme in the domain so IPv6 is tend to be moved to bottom of their priority list.

- There are multiple IPv6 protocol specifications (not the basic ones) that need to be finalized, or revisited / augmented. Also, vendors need to support those auxiliary specs.
- For instance, routing protocols for IPv6 (RIPng and OSPFv3) do not have authentication mechanisms on their own - the specs just say "use IPsec". However, it is not enough to say "use IPsec", we need the gory details on how we should secure routing protocol exchanges by IPsec. Most of the routing protocol exchanges are made with link-local scoped multicast, and there is no good way to automate key exchange for multicast.

We are now very optimistic about IPv6 deployment worldwide. Now the US government requires IPv6 support in their purchase requests to big ISPs (like DARPA-funded networks), there are multiple European projects for IPv6 backbones, and there are a lot of Asian initiatives for IPv6 deployment. We Japanese needs to keep up the current pace, cooperate with other countries, and try to lead the IPv6 deployment worldwide.

Future plans

As described above, we are still implementing and testing various internet protocol proposals at KAME, and try to keep *BSD IPv6/IPsec stack up-to-date. It is already decided that the KAME project will continue until spring 2004.

There are multiple requests to continue the effort beyond spring 2004, so that we can have a single point of contact for IPv6/IPsec-related issues in BSDs. For instance, some of the vendors are worried how they can support newly developed protocols after the termination of KAME. If you have any commenst on this aspect, please contact secretary@kame.net.

Conclusion

This paper talked about the history and status of the KAME project, and IPv6 deployment status in Japan as well as worldwide.

For details of the KAME IPv6/IPsec implementation, you can just browse the source code (our code is heavily commented with references to RFCs), or refer to multiple papers presented in various occasions [4].

Author's address


```
Jun-ichiro itojun HAGINO
Research Laboratory, Internet Initiative Japan Inc.
Takebashi Yasuda Bldg.,
3-13 Kanda Nishiki-cho,
Chiyoda-ku, Tokyo 101-0054, JAPAN
Tel: +81-3-5259-6350
Fax: +81-3-5259-6351
Email: itojun@iijlab.net
```


Bibliography

[1] S. Bradner and A. Mankin
IP: Next Generation (IPng) White Paper Solicitation
RFC1550 (December 1993)
`ftp://ftp.isi.edu/in-notes/rfc1550.txt`

[2] S. Deering and R. Hinden
Internet Protocol, Version 6 (IPv6) Specification
RFC1883 (December 1995)
`ftp://ftp.isi.edu/in-notes/rfc1883.txt`

[3] S. Deering and R. Hinden
Internet Protocol, Version 6 (IPv6) Specification
RFC2460 (December 1998)
`ftp://ftp.isi.edu/in-notes/rfc2460.txt`

[4] Jun-ichiro Hagino
Mbuf issues in 4.4BSD IPv6/IPsec support (experiences from KAME IPv6/IPsec implementation)
USENIX Freenix2000 (June 2000)

Technical Solutions for Controlling Spam

Shane Hird
Distributed Systems Technology Centre
Level 12, S Block, QUT Gardens Point
Brisbane Qld 4001, Australia
email: shird@dstc.edu.au

Abstract

As the commercialisation of the Internet continues, unsolicited bulk email has reached epidemic proportions as more and more marketers' turn to bulk email as a viable advertising medium. Concern about the proliferation of unsolicited bulk email, or spam, has continued to grow, with the Internet community increasingly turning to both regulatory and technical solutions to alleviate the problem. While marketers seek to reach a larger and larger audience as an attempt to increase their returns, consumers are seeking effective means to avoid being targeted. There is already a broad range of counter measures to deal with the problem of spam, some of which have been successfully deployed in commercial environments. This paper will attempt to evaluate some of the existing technical solutions to control the ever-increasing volume of unsolicited bulk email.

Introduction

The prevalence of unsolicited electronic mail, or spam, has steadily become a significant problem for network administrators, service operators and Internet users in general. Recipients of large quantities of unwanted mail find it time consuming or difficult to differentiate desired mail from spam, reducing their productivity. Aside from the direct costs generated by the consumption of Internet resources, such as network bandwidth, processing, storage space and other requirements, there are also many indirect costs produced as a consequence.

Past experience has taught users to be reluctant to give their addresses out for fear of being added to and traded among thousands of mailing lists. This behaviour restricts businesses from acquiring addresses for legitimate use. Likewise, a business may be reluctant to email their customer base for legitimate purposes for fear of being perceived as spamming. Also, the technical measures being used to prevent the relatively few that abuse the infrastructure result in compromises that must be endured by everyone.

Although distinguishing spam from legitimate email is subjective, majority agree that all forms of unsolicited email are undesired. People who receive relatively small quantities of spam however, will often accept it as an annoyance and tolerate the problem. However this attitude is one that can contribute to allowing spammers to continue their abuse, by tipping the economics in their favour. With the cost-shifting associated with email, and the distribution of costs over such a wide base, the costs to the spammers are almost none, and although the costs are shifted to the end recipients, they are also relatively negligible. The fact these individual costs are so small is what creates a problem on the larger scale. Noble prize winning Ronald Coase hypothesized that it is especially dangerous for the free market when a business that cannot bear the costs of its own activities, distributes those costs to the population at large.

What makes this situation so dangerous is that when millions of people each suffer only a small amount of damage, it often is more costly for each individual victim to recover the minor damages imposed upon them. The population will continue to bear those unnecessary and detrimental costs unless and until their individual damage becomes so great that those costs outweigh the transaction costs of fighting back. Hence, spammers are able to continue their cost-shifting form of marketing.

This paper holds the view that any form of unsolicited bulk email (UBE) is considered undesired spam, inclusive of unsolicited commercial email (UCE). This is mainly due to the similarities between the two in terms of cost-shifting and the difficulties in a technological solution for one and not the other. Also, blocking only UBE and not UCE will potentially open a loophole for spammers to send UBE rather than UCE, even though the indirect result of the UBE may be commercial in nature.

There are three general categories to addressing the problem of spam: informal measures, such as social norms and self-regulatory efforts; technical measures, which will be the main focus of discussion; and legal responses, both existing and new legislation to address the rising problem.

Spamming Activity

To understand the issues involved in controlling spam, the methods employed by spammers should be investigated. The basic activities of most spammers are briefly outlined below.

Harvesting Addresses

Due to the low response rate of advertising through unsolicited email, it is important for a spammer to have a comprehensive list of email addresses. Because few people would be prepared to knowingly hand over their address to a spammer, addresses are usually collected from the public domain. Common methods and locations spammers use for automatically harvesting addresses include;

- Posts to UseNet with your email address.
- Mailing lists
- Web pages (especially guestbooks and forums)
- Various web and paper forms
- Domain contact points
- Dictionary attacks on both username or domain
- Predictable email address patterns
- From white and yellow pages (eg. Bigfoot)
- Chat rooms

Addresses are usually harvested using automated tools that analyse online content for patterns matching that of an email address. Such tools may also include the ability to search web sites and newsgroups using a particular keyword, making use of existing search engines. This can make it possible to more accurately target users interested in a particular area.

Account Hopping

As the sending of unsolicited email is considered an abuse of network resources, many service providers prohibit the activity as part of their terms of use. As such, spammers may find that their accounts are continually terminated as users report their actions. To prevent this, spammers often make use of free trial accounts, 'spammer friendly' ISPs, or stolen account details. This poses a particular problem for technical solutions such as blacklisting, because the abusive users are sharing the same network and mail server as many other legitimate users. In this situation, a lot of the responsibility falls on the ISP to ensure their users abide by their terms of use.

Composing

The low response rate from unsolicited email advertising, and the amount of other spam entering a users inbox, requires a spammer to compose messages that are more likely to capture the users attention. Users have become accustomed to manually filtering spam from their mail, and will quickly delete messages that appear to be spam just from the subject line without even reading the message. This has encouraged spammers to entice users into opening mail or visiting web sites by including seemingly legitimate subject lines and message bodies.

Increased use of automated filtering tools has forced spammers to try and avoid certain keywords and phrases that are included in majority of spam. Commonly seen methods to bypass content filters include substituting numbers for letters (i.e. zero instead of 'O'), using superfluous spaces or other symbols (i.e. 'w 0 r k f_r_0_m h 0 m e!!') or inserting random hidden comments in the case of HTML messages.

As collaborative filtering becomes more popular, spammers are finding it is increasingly difficult to send exactly the same message to hundreds or thousands of recipients. To counter this, unique identifiers are included with each

individual message so they generate different checksums, otherwise known as 'hash busting'. These identifiers are usually in the form of random strings appended to the subject line, random or personalised messages in the body and random hidden comment tags in HTML messages.

Sending

With a list of addresses and a message composed to send, a spammer will use one of the many bulk email tools available to get his message across. To avoid getting his account terminated, attempts are usually made to hide the point of origin. This is commonly achieved by making use of misconfigured servers such as open relays or proxies. Not only does this help to hide the origin of the spam, but also by offloading the responsibility of delivering mail to an open relay, a far greater throughput can be achieved. It will also help to deflect complaints to the relay.

Spammers will also commonly make use of vulnerable CGI scripts such as greeting card sites or feedback forms to send email without revealing their origin. Alternatively, if no open relay or other intermediary system can be found, or have been blacklisted by other sites, many bulk email tools have the ability to connect directly to the user's mail server to deliver the mail directly. Because a mail server cannot distinguish between a legitimate mail relay and a spammer with bulk email software, this can be difficult to blacklist, but does reveal more of the spammer's origin.

Technical Measures

Currently the most effective and commonly used means of controlling spam is through technical solutions. A variety of methods already exist, each with its respective merits and disadvantages.

Blacklisting

Probably the most common method of blocking spam is rejecting connections at the mail server based on the origin. The usual and supported method of achieving this is done by taking the IP address of the remote mail server, or dialup user, converting it to a domain name using the ip4r format and querying a "DNS zone" which lists blacklisted addresses (i.e. a.b.c.d becomes d.c.b.a.lookupzone.com). Depending on the DNS-based blacklist database used [1] if the address is listed the result returned would be the loopback address (127.0.0.1) with the last octet modified to indicate the type of record found.

The different DNS based blacklisting services have varying addresses and networks listed, based on their purpose and policies. Common databases include open proxies, open relays, networks or individual addresses guilty of sending spam, networks known to consist of dial-up users, and various other less common lists. Most of the lists contain networks that mail server operators are unlikely to want connecting to their server. For example, dial-up users should only be sending mail through their own ISP's mail server, and not connecting directly to the receiving server, such as spammers often do when they cannot use an open relay. So connections from dial-up users should be rejected unless they are from your own network. It is however up to the subscribers of these services to decide what action to take when a connection is made from a network listed in these databases. The usual action is to simply reject the connection at SMTP time, with an error indicating the database the network was listed in.

Subscribing to these lists requires a high level of trust to be placed in the maintainers, due to purposely refusing mail from networks which others have considered irresponsible. Entries are made into the lists using methods and policies that vary from list to list, though generally nominations are made for a particular network or address. Moderators then investigate the network and attempt to contact the owners to correct the problem. If the problem isn't fixed, and nominations for the network continue, the address range may be added to the list. These lists are widely used by mail providers, which provides a strong incentive for ISPs to ensure their users don't abuse the network and get the entire ISP's network or mail server blacklisted.

Complaints are a major deterrent for spammers. Service providers are determined to keep their networks off the blacklists, to keep their other customers content and maintain their image. For larger ISPs with policies against sending UBE, complaints from recipients will often result in termination of the users account. The true sender of a spam message can be difficult to determine, as the senders often attempt to hide their true origin. There are various products and services [2] that can automate the process of determining and sending complaints to the spammer's service provider. Unfortunately, complaints often fall on deaf ears, and nothing is done to address the problem. This may lead them to be eligible for nomination on the blacklist services.

Because blocking based on origin occurs before a message is received and processed by the receiving mail server, blacklisting can help reduce many of the costs associated with UBE. Although blocking known and potential channels of abuse can prevent a large amount of unsolicited mail with a minimal amount of resources, it isn't a completely effective solution, and cannot filter spam that comes from unlisted servers. Like any filtering system,

blacklisting also presents the possibility of eliminating wanted messages, especially when an ISP's network gets listed because of an abusive user or a misconfigured mail server.

Spammers can establish an account with an ISP that has not been listed in the blacklist for being 'spammer friendly', and relay their messages through the ISP's mail relay like any other customer. For an ISP wishing to remain out of the blacklists however, this will usually result in the spammer's account being terminated, but not before the bulk mailing has reached its intended recipients. To aid against this problem, there is also a separate blacklist, 'Spam Whack', which helps ISPs identify subscribers who have been terminated by other ISPs for spamming. [3]

Whitelists and Channels

The concept of established channels to restrict unsolicited content (most notably spam) has long been used in instant messaging applications. These clients will usually allow you to specify that you should only accept a message from someone who is in your list of contacts. [4] This allows you to have a list of pre-approved contacts that are able to communicate with you. These whitelisting methods can also be applied to devices such as mobile phones should text messages through this medium become as problematic as spam through email.

This whitelisting method can also be applied to email, by rejecting messages that don't come from an already known contact or from a trusted domain. The most obvious disadvantage of such a method being that it restricts communication to already established contacts, which is impractical for the majority of end users. It is an acceptable method for instant messaging environments, where contacts are generally made through other mediums first. Email however, is often used as a medium for establishing new contacts.

A variation of this approach is to use an automated challenge-response system, which is used to verify that the sender's account exists, by sending a challenge to the sender and requiring a valid response. A valid response usually results in the sender being whitelisted so that the 'handshake' isn't required for future communication. Existing implementations of this system are the 'Tagged Message Delivery Agent' (TMDA) [5] and 'Active Spam Killer' (ASK) [6]. Both of these systems effectively quarantine any message sent from an address that isn't whitelisted, and automatically reply with a request for confirmation message. This must be replied to in order for the message to be released from containment and presented to the user. If the message is confirmed, the address is whitelisted so that future messages from the user needn't be confirmed.

Automatic replying or responding is seen by many to be a bad practice, with majority of spam having forged 'From:' addresses, it only serves to multiply the bandwidth already wasted by spam. The forged addresses are sometimes those of innocent victims, who are then bombarded with bounced mail. This is already apparent due to the number of invalid addresses in the lists used by spammers, causing a large number of the messages to bounce by the relaying MTA anyway. In ill configured systems, automatic replying may also cause dangerous mail loops, especially in the case of mailing lists.[7] There is also the possibility for different kinds of abuse, such as signing people up to mailing lists, with the automatically generated reply being considered confirmation. [8]

It also may not always be possible to 'handshake' to establish a communication channel, in the case of temporary or unattended mail accounts, such as order confirmation messages from online purchases. It also slows down the overall email process, especially for dial up users. They send the original email, check their email again at a later date to receive the 'challenge' message, and then send the response. This may take than 24hrs for some people, depending on their usage.

Because the 'mailer-daemon' address should always be whitelisted, to receive notifications when an email address is invalid when legitimate mail is sent out, spammers may also start using this address to bypass the system, although there are measures to prevent this being a problem. Spammers are also already using addresses from the same domain, in the hope they will either be whitelisted or at least closer attention will be paid to the message. For this reason, the network the sender originated from should also form part of the whitelist information. The extra overhead of a challenge-response system will also be seen as an annoyance to many people, who may choose not to bother sending the confirmation message, or bother communicating with you any further. The net effect is people will still be required to sort through their 'quarantine bin' for unconfirmed legitimate messages, reducing the effectiveness of the solution.

An alternative, though drastic, solution is one that makes use of a global implementation of PKI, with the use of digital signatures on email. A user can then filter based on these signatures, with the ability to easily blacklist individual senders; offenders could also be more easily traced and dealt with accordingly. Unfortunately this centralises an already distributed system, going against everything the original infrastructure was designed to accommodate. It would place significant restrictions on senders, who would have to obtain certificates, which may discriminate users who cannot obtain such a certificate. The use of referral networks or a 'web of trust' could alleviate the problem of centralisation; however many other problems still stand, and spammers will undoubtedly not have much trouble obtaining many certificates to use for spamming. It would also require adoption by a critical mass to be effective.

An alternative to filtering based on the sender's address is to filter based on the recipient's address. This can be achieved by having multiple accounts, distributing different ones to different contacts. This includes using different

addresses to sign up for different mailing lists or whenever a valid email address is required. Having mail from different senders coming through separate channels, allows a user to apply different levels of protection, or completely ignore, channels that are experiencing higher levels of unsolicited mail.

Although this can be achieved by having multiple aliases for an address, or actual accounts, an alternative approach is to make use of 'disposable address' services, such as 'Spamgourmet'. [9] These services are implemented in a number of different ways, one method being to allow aliases for an address to be assigned a fixed number of messages that should be allowed through. Any message received after that amount is considered spam and instead of being forwarded to the real account, is used for collaborative filtering purposes.

Disposable addresses allow for separate channels of communication which can be terminated if the signal to noise ratio becomes too high. People who adopt such forms of communication may find that they start to receive legitimate mail on their 'disposable' addresses and would be unwilling to revoke those channels for fear of losing desired correspondence. There is still a benefit from having mail from different senders sorted into separate channels though, and such a system is useful in instances where it is known that only a fixed number of messages should be received, such as for confirmation messages when signing up for online services.

Spam Poisoning

People who receive the least spam are typically those who have kept a low profile in terms of keeping their address from being publicly exposed. Restricting the distribution of one's address to only trusted parties, effectively 'hiding' from the spammers, is an effective means of reducing spam. It is however impractical for those wishing to be open to anonymous correspondence.

To prevent addresses from being harvested, yet still published to the general public, people will often 'munge' their address. This typically involves disguising an address in such a way that it is readable by humans, though software designed to parse addresses would interpret it incorrectly or not at all. This can be achieved by swapping or inserting words within the address, and including instructions on how to unscramble the address (eg. '`user@ exampleREMOVETHIS.com`'). Other methods include displaying the address using an image, or using script that generates the address to be displayed at the client side.[10] Recent drafts of the Usenet message format RFC specify that the 'From:' line of a newsgroup posting must contain either a valid email address or an email address ending in ".invalid". Your munged email address should really comply with this forthcoming standard (e.g. user@REMOVE-CAPS-AND-INVALID.example.com.invalid).

Although this method can greatly reduce the amount of spam received, particularly as majority of spam is addressed to 'fresh' addresses harvested from places such as Usenet, it's not without its drawbacks.[11] Some spammers now have harvesting software that can remove widely used munges like "NOSPAM". It also places a burden on people needing to unscramble your address, and to those systems whose addresses may have been used in forgeries. Once your email address is revealed just once, either by mistake on your part, or through the process of 'list cleaning' or 'guessing', all efforts expended trying to conceal the address were wasted. The address is often permanently added to many other lists and traded amongst spammers.

Another known method to prevent email addresses from being harvested is to pollute the areas being trawled with numerous false addresses. This aims to reduce the signal to noise ratio for spammers to a point that it either completely discourages them from harvesting addresses altogether, or requires them to manually collect email addresses. Preferably this would be done through an 'opt-in' scheme, which could also result in increased accuracy in their direct marketing.

People doing the harvesting are usually only doing so to sell lists to spammers, so are often more concerned with the number rather than the accuracy of their collected addresses. The spammers buying and using the lists commonly use a false 'From:' address and are oblivious to any bounced messages. Their only indication that a list of addresses is badly 'polluted' could be the lower response rate, which would usually be quite low and sporadic anyway. 'Web-bugs' and other verification means are increasingly being used however.

Examples of automatic 'spam poisoning' systems are WPoison [12] and Sugarplum.[13] Rather than a static list of fake addresses posted to a web page, these systems will dynamically generate an unlimited number of random fake addresses through the use of CGI applications for harvesting programs to collect. The pages generated will also contain hyper-links to seemingly different pages for the harvesting spiders to follow. The links however, link to the same CGI program to generate yet another page, trapping a harvester into an endless loop of collecting invalid addresses.

Harvesting 'spam bots' have been developed to detect addresses that don't belong to a valid domain, or pages that contain nothing but email addresses. To counter this advancement, these systems will also insert other random text and links, as well as make use of dictionary words to make addresses seem valid and less random. This is helped by the fact that nearly every word in the dictionary appended with '.com' is a registered domain name.

Aside from completely fictitious addresses, such systems can also be configured to generate known 'teergrube' addresses.[14] These are especially set up addresses on deliberately crippled mail servers that are able to hold open

a connection for prolonged times, substantially slowing down any spammer which runs into such an address. This technique is typically coupled with blacklisting so that only blacklisted hosts, which connect to the mail server, are slowed down.

Address harvesting programs are evolving however, and have grown wise to such techniques. Most sites that utilise some form of spam poisoning, will usually have a human readable note describing the system. The harvesting programs will often search for such warning labels, and avoid such sites, though this does have the benefit that other genuine addresses on the site aren't harvested. Increasingly though, such programs are using search engines to go to pages directly instead of following nested links. They typically search for such phrases such as 'guestbook' or 'forum', that are likely to have many legitimate addresses, then harvest the resulting pages. This avoids being caught in traps or indexing dynamically generated content, and lets the search engine do most of the hard work.

Collaborative Filtering

For most users, the problem of spam is dealt with in part by their destination operator, the provider of their email account, which is typically their ISP or another third party email provider. The use of collaborative filtering can be quite an effective means of blocking spam for these operators. It particularly excels in the ability to detect messages being sent to multiple recipients. With a large number of participants, they have access to a large message base to analyse and detect bulk mailing patterns.

Two systems that exploit the fact that spam usually consists of exactly the same or very similar messages being sent to multiple recipients is Vipul's Razor [15] and the Distributed Checksum Clearinghouse (DCC).[16] Both DCC and Razor are distributed, collaborative, bulk mail detection and filtering networks. When a user or 'spam trap' address receives spam, the message is hashed into a unique identification of the spam that is then submitted to the closest Razor server. In the case of DCC, all messages received are treated this way and the server keeps track of the count of submissions, which it shares with other DCC servers. Using this mechanism, DCC and Razor establish a distributed and constantly updating catalogue of bulk mail in propagation, majority of which is spam. Clients that make use of these services can then hash received messages and check them against the Razor or DCC databases.

This system is however open to abuse from people who submit hashes of legitimate mailing list messages, either deliberately or unintentionally through an automatic process. Like other spam solutions that have problems with mailing lists, this can be overcome with the use of whitelists for sources that shouldn't be flagged as spam. A common method of circumventing collaborative filtering involves modifying each message with one or more unique tags, so that a different hash would be generated as a result. The DCC system does however employ 'fuzzy' checksums, which are designed to only ignore differences that do not affect the meaning of the message, particularly in English. There is a limit to the effectiveness of such fuzzy hashes without the risk of false positives however, so this approach may be somewhat limited given advancements in future spamming techniques.

An alternative to community-managed systems with their associated problems is a commercial service such as Brightmail.[17] This service is similar to Razor, although without submissions from the public. It utilises a 'Probe Network', which is a collection of email addresses (with a statistical reach of "over 100 million mailboxes") planted throughout the Internet to be harvested by spammers. The mass of spam caught by these decoy addresses is then monitored in real time by a full-time staffed centre at Brightmail, which generates and writes rules to block the spam, which are distributed to Brightmail customers for their use by Brightmail managed systems.

Although seemingly quite an effective system, there are costly license fees involved, and is impractical for individuals or small ISPs. Brightmail anti-spam software is used at AT&T Worldnet, Critical Path, Hotmail, Excite@Home and Motorola, all of which are major email account providers. Despite the elegancy of such a system, the amount of spam that manages to slip past Hotmail's filter is evidence enough that this is far from a 100% effective solution. Most users of DCC and Razor report higher success rates using those public services than with Brightmail.

Content Filtering

Content filtering using heuristic systems can help alleviate the problems caused by legitimate bulk mail using other technical solutions, as mail is filtered based on the nature of the content, rather than the channel through which it arrived. It can also be implemented transparently, without requiring end users to change their behaviour or client software. For this reason, it is a common method implemented by many destination operators, particularly to reduce UBE that is commercial or offensive in nature, which is most likely to contain predictive keywords that can be used for filtering.

By placing a filtering system on the server side, it allows users to have filtering without the need for any client side software; it also allows the flexibility of allowing users to only download messages that haven't been tagged as spam. However it requires a large amount of resources on the server for processing all mail for all accounts, and creates a difficulty for tailoring the filter to each specific users needs. Content filtering also does little to address the bandwidth and storage capacity problems caused by spam, as the message must still be received to be processed.

It also presents the problem of what to do with email that gets flagged. Simply discarding flagged messages is considered a bad practice, mostly because of the implications of false positives.[18]

It may be acceptable to reject the message if the score obtained is exceptionally high and very unlikely to be a legitimate message. If this behaviour is desired, the best option is to reject the message at SMTP time, with an appropriate error message. Not only does this minimise the amount of resources consumed by the unwanted message, but it will also provide an immediate rejection message to the MTA, which will propagate to the user sending the message. If it is indeed a legitimate message, the user will be aware that the filter rejected it and are given the chance to reword their message. If the message was sent from a bulk email program without using a relay, it is possible the address will be dropped from their list upon encountering the error.

Experience shows that content filtering doesn't currently, and is unlikely to ever, achieve 100% accuracy. Users would rather have a filter that misses a small percentage of spam (false negatives) rather than a filter that incorrectly identifies a small percentage of desired mail as spam (false positives). This risk of false positives means a conservative approach to filtering should be taken. Filtering solutions generally do not delete tagged mail, but deal with it in such a way that it is not disruptive to other mail which passes cleanly through the filter, however is still accessible for review and possible retrieval.

The common method of achieving this is to add headers and tags to a message so that a user may filter the messages at the client end. This may pose problems for individuals who download their email using a modem, and must still wait for unwanted messages to arrive. In this case, the main concern caused by spam has not been eliminated. This can be alleviated by email systems that allow clients to preview email headers, so that they can discard or ignore messages before being fully downloaded, or through using an IMAP mail service. Another solution is to generate a daily digest of caught spam, in the form of a short extract of each message, or with just the subject and 'From:' address. If an important message was noted in this digest, the message could be retrieved by a web interface from the email provider. The same web interface could also be used for managing configurations such as whitelists and threshold limits.

Fast gaining in popularity, though still in its early stages of development, is the open-source content filtering solution SpamAssassin.[19] This project is being developed by a handful of developers and a vast array of contributors, as is typically the case with open source projects. With this type of application, this is a particularly effective development strategy. People have an incentive to contribute improved filtering methods that help catch the spam they are seeing pass through the filters, but that perhaps others aren't. This is apparent by the active mailing lists, which also suggest there is an increasing number of deployments of the product, including commercial environments.

Rather than flagging messages if they contain any single particular known phrase or characteristic, SpamAssassin uses a weighted scoring system. This allows SpamAssassin to depend on a variety of different tests, each of which can be assigned a different weight, including negative weights. These tests not only involve textual patterns to be detected in the content of the message, but also can involve tests such Razor and DCC checking (collaborative filtering), detecting invalid or suspicious headers, number of recipients and others.[20] If a sufficient score is acquired in analysing a message, one that exceeds a custom configured threshold, then the message is considered spam and SpamAssassin will tag the message appropriately, which can then be used to process the message as desired. Usually this involves re-writing the message so that the subject easily identifies the message as spam, and the body contains a summary of the tests which were flagged.

To arrive at the values for the different weights for the tests, SpamAssassin uses a 'genetic algorithm'. Essentially this takes a collection of both spam and non-spam messages, and adjusts the weights of the rules accordingly. Rules that occur mostly in the spam body of messages, and occur frequently, are weighted heavily, while rules that tend to occur in both message stores are weighted less. This has the effect of minimising false positives, while also minimising false negatives. Recent documentation states that out of a 257,000 message corpus, there were 140 false positives and 3537 false negatives, making it 98.57% accurate. This was achieved using only content analysis, without collaborative checksum, blacklist or automatic whitelist checks, which would likely improve the accuracy even further.

The mechanism that SpamAssassin and other similar filters employ is similar to how a real person would assess whether a piece of mail was spam. A person would look at the 'From:' address and subject, see if it's from someone they know or something they're expecting, or if it looks randomly generated or commercial in nature. Passing that, a person would quickly scan through the content of the message, establishing evidence about the nature of the message. Once that evidence exceeds an acceptable threshold; the message is deemed spam and dealt with accordingly. If it is considered legitimate, SpamAssassin can also make use of automatic whitelisting (AWL), which allows the system to keep a credibility record of a particular address. This record can then be used in future to score the message based on the amount of legitimate mail sent from that particular address.

Filtering alone cannot be considered a complete solution; spammers are able to work around the filter by running their message through the filter, mutating it until it manages to pass through, analogous to virus authors that modify their creations until the heuristic engines don't flag them as suspicious. As with filtering tools for spam though, existing anti-virus tools still manage to detect a large percentage of unknown viruses, even though they are readily available to be tested against. This suggests that even with filtering tools in place, and some spammers evolving

to circumvent them, they are still useful in blocking a significant proportion of spam. This is further emphasised by different deployments having different thresholds and different tests, which spammers would be required to continually test against.

Payments

A method to reverse the 'cost-shifting' that occurs with email is to enforce a payment for mail sent, which would produce a sender pays rather than receiver pays environment. Requiring advertisers to pay for the messages they send would potentially reduce the amount of unsolicited mail, as sending out literally hundreds of thousands of messages becomes prohibitively expensive, even if each individual cost is small. For the average user however, personal correspondence involves relatively far fewer messages, so the cost could be reasonable. It may also be possible for the receiver to refund payments to the sender if the message was desired, and also whitelist individual contacts such that future correspondence doesn't require payment.

This electronic postage approach could require substantial overhead costs, some degree of centralisation, co-operation and widespread adoption by many users and ISPs. It would also probably be met with a high level of opposition from users who are currently able to communicate for 'free' using the existing infrastructure. Traditional payment systems that are traceable are not appropriate where privacy must be maintained, there are however a few anonymous electronic cash systems.[21] Current electronic payment systems have had little success in the real world though, and it is unlikely current services could be capable of the millions of payments that would be required for use in email.

An alternative to senders making monetary payments is one where senders are required to perform time-consuming computations.[22] Although these computations would be free to perform and could be evaluated in a reasonable time, the time required for mass mailing would be prohibitively long. The obvious advantages of such a system are that it would be free and could be implemented to still allow anonymity by not requiring any centralisation. Like other spam controlling solutions, it would present a problem for legitimate bulk email, although once again this could be alleviated with the use of whitelists. With the use of automatic whitelisting, mailing lists may only be required to 'pay' for the first few messages sent to new subscribers. It could be assumed that existing subscribers have already automatically whitelisted the mailing list, and don't require further payment.

Various schemes have been proposed to implement a computation payment system; HashCash [23] and CAM-RAM [24] are two known implementations in development. The computation, or *pricing function,* used by the payment system should be made to be arbitrarily expensive to compute, but possible to verify almost instantly. HashCash makes use of n-bit partial hash collisions on chosen texts, where the chosen text is something unique to the recipient, usually their email address concatenated with the current date and a hash sum of the message body. The more bits of collision required, the more time it takes to find a hash which satisfies the required number of collision bits with the hash of the chosen text.

By requiring that the chosen text contain the user's email address, it prevents bulk mailers from using a single generated *hashcash token* for many different users. Including the current date and a hash sum of the message in the chosen text, prevents people from using the same *hashcash token* to send mail repeatedly to the same user, or 'double spending' a token. The token is included in the headers of the email message, such that recipients can verify the validity of the tokens and optionally reject messages that don't comply. With n-bit partial hash collisions, it is possible to require arbitrary amounts of collision bits, which can adjust the 'expense' of the computation. This would allow a user to require more collision bits in the tokens as a means of increasing the 'cost' of postage, if unsolicited or superfluous mail still remains a problem.

A complication with this solution is the difference in processing power between systems would allow some users to generate tokens faster than others. To prevent spammers from just using faster hardware, the complexity required for the computation should be benchmarked against a modern machine or specialised hardware. Given this complexity requirement, some users may not be able to generate a token in a reasonable time given their available resources. A solution could be for their ISP's mail relay to generate the tokens for them, assuming the ISP has the resources to generate such tokens within an acceptable time. The mail server should only generate this token when authentication is used to relay mail, and a token isn't already included. The ISP may possibly charge the user for each token generated or allow a fixed number of free tokens. Only generating the token when authentication is used would prevent spammers from directly connecting to the server for local mail delivery and have a token generated for them.

Ideally, the client system rather than the mail server should do the processing. To avoid needing to modify all existing clients to support the generation of tokens, a generic proxy could be developed for each platform that could intercept mail and generate the required tokens, as well as be used for verifying tokens during mail retrieval. This would be compatible with all existing mail clients, and would be otherwise transparent. In the case of 'webmail' style services, it could be possible to use Java applets or some other client side execution mechanism to generate tokens to avoid placing load on the server.

Even in a 'sender-pays' environment, existing paper-based junk mail has shown that advertisers are still willing to pay to get their message across. Enforcing payment from the sender may place the law in favour of the spammer, as he has legally 'paid' for the message to be delivered and it may be illegal for a service provider to block such messages. This means the end result of enforcing a payment system could be spam that is more accurately targeted, but cannot be legally filtered by any upstream provider. This may in fact cause more unsolicited mail for the end user; however a non-anonymous payment system would allow end users to blacklist repeat offenders more effectively at the client end.

If a payment system were to be implemented, instant global adoption may be required to prevent people from losing desired correspondence from people who cannot easily comply with the new system. However a payment system could be incrementally introduced by combining it with other solutions, such as filtering and blacklisting, by having messages that conform to the payment system being biased towards not being flagged as unsolicited mail. All mail would still be subject to content analysis and blacklisting as normal, though messages that don't conform to the payment system would be given less credibility. This would also mean senders aren't paying for the message to be delivered, but instead for a higher priority rating when filtering is done. The net effect would be reduced false positives, with the ability to decrease the threshold of the filters to also decrease false negatives, as more people adopt the payment system.

Opt-out Lists

Individuals that do not wish to receive spam have the option to include their address on an established 'opt-out' list, or request to be removed from existing mailing lists. Opt-out often refers to email advertising lists in which recipients are signed up without their knowledge or permission, but may request to be removed from the list. There is little evidence that spammers use these lists to clean their own lists however, instead they are usually used to verify that a given address exists. Opt-out lists also violate the principal that all communications should be consensual. A better option for bulk mailing, used by legitimate lists, is the concept of 'opt-in' lists. With existing legislation and technology, this is a difficult system to enforce.

Internet Mail 2000

With the problems apparent with the existing email infrastructure, an alternative to trying to add extensions to the current protocol is to create a new protocol from scratch. 'Internet Mail 2000' is one such idea, proposed by D.J Bernstein, the creator of qmail among other things. The current email infrastructure is based on a 'push' mechanism, where the entire contents of an email is replicated for each recipient, and the effort of addressing N targets instead of just 1 is near zero. This leads to the recipient bearing the bulk of the costs, as it is their bandwidth and local storage resources that are being used.

IM2000 attempts to overcome this issue by effectively implementing a 'pull' mechanism, which is based on the idea that mail storage is the sender's responsibility. The project is built around D.J Bernstein's concept proposal, which outlines some of the ramifications of the new infrastructure: [25]

- Each message is stored under the sender's disk quota at the sender's ISP. ISPs accept messages only from authorized local users.
- The sender's ISP, rather than the receiver's ISP, is the always-online post office from which the receiver picks up the message.
- The message isn't copied to a separate outgoing mail queue. The sender's archive is the outgoing mail queue.
- The message isn't copied to the receiver's ISP. All the receiver needs is a brief notification that a message is available.
- After downloading a message from the sender's ISP, the receiver can efficiently confirm success. The sender's ISP can periodically retransmit notifications until it sees confirmation. The sender can check for confirmation. There's no need for bounces.
- Recipients can check on occasion for new messages in archives that interest them. There's no need for mailing-list subscriptions.

With these ideas in mind, several aspects need to be addressed by the protocol implementation. Various prototypes for a new protocol implementing these concepts have been proposed and are described on the IM2000 project page.[26] At the time of writing however, the project has been fairly dormant. A global deployment of an implementation is unlikely anytime in the near future.

Despite the advantages of a system which implements aspects of the IM2000 concept, many of the ideas also present significant complications, and would also require global adoption to be truly effective in preventing spam. Although the protocol is being designed essentially from scratch, the proposed solutions provide quite complicated solutions to admittedly difficult problems, given the requirements. Most proposed prototypes also require some level of centralisation, and lack the ability for anonymous communication. Although this feature in the current system leaves it open to some level of abuse, it is also a necessary requirement for many. Despite the problems with the existing infrastructure, it is a reasonably robust protocol, which poses a significant challenge to switching over a critical mass to a more complicated and somewhat more restrictive system.

Conclusion

A problem with the success of defensive technical solutions is they hide the extent of the real problem from the end user. The destination operator (i.e. an ISP) is still forced to endure a significant cost for fighting the problem, which ultimately end users must compensate. Ideally spam should be stopped before it takes place. Defensive technical measures that reduce delivery rates may ultimately achieve this, however it would take a significant reduction for a significant proportion of end users. Shielding end users from the extent of the problem could be preventing them from pressuring for more offensive measures to be taken, which could stop the problem of UBE from the source.

An approach for preventing spam at the source is through regulatory measures, although this has not been the focus of discussion. There has been some amount of success with legislation addressing the most extreme instances of spamming, and a number of jurisdictions have enacted specific laws in an attempt to regulate spam.[27] But current legal approaches seem to have been no more successful than technical responses, with the only result being a great deal of uncertainty surrounding the legality of spam.

The current state of the art in spam prevention involves the collective use of several of the discussed solutions. An effective weighted scoring content filter, combined with collaborative filtering, blacklisting and whitelisting has shown to be highly successful in stopping the majority of spam, transparently and with minimal negative effects. The SpamAssassin filter combines all of these techniques together, and is generally regarded an effective packaged solution. The relentless efforts of 'anti-spammers' in reporting abusers will also continue to play an important role in stopping the abuse getting out of hand, as will the participants in collaborative filtering efforts. As spammers evolve to circumvent existing measures, thresholds may be tightened with more tests added. Other solutions such as 'HashCash' payments, which can be incrementally introduced, could also be included into the scoring procedure.

With the current state of the email infrastructure, a 'magic-bullet' technical or regulatory solution is unlikely to be possible. Ultimately, a consensus approach that coordinates legal and technical responses is likely to provide the only satisfactory solution. Without an effective solution, spam can be expected to be here to stay, and grow to decrease the value of what is otherwise an efficient and invaluable communication medium.

References

1. For a complete list of DNS-based spam databases – *DNS-based Spam Databases* (http://www.declude.com/JunkMail/Support/ip4r.htm)

2. SpamCop spam reporting service, *SpamCop.net* *(http://www.spamcop.net)*

3. *Spam Whack!* (http://www.spamwhack.com)

4. ICQ Inc., *Security and Privacy – Anti Spam* (http://www.icq.com/support/security/spam.html)

5. Jason R. Mastaler, *Tagged Message Delivery Agent* (http://software.libertine.org/tmda/)

6. Marco Paganini, *Active Spam Killer* (http://www.paganini.net/ask/)

7. D.J. Bernstein, *Tools in the war on mail loops* (http://cr.yp.to/proto/mailloops.txt)

8. K. Moore, *Recommendations for Automatic Responses to Electronic Mail* http://www.ietf.org/internet-drafts/draft-moore-auto-email-response-00.txt

9. Spamgourmet, *Spamgourmet - disposable email addresses* (http://www.spamgourmet.com)

10. W.D Baseley, *Address Munging FAQ: Spam-Blocking Your Email Address* (http://members.aol.com/emailfaq/mungfaq.html)

11. Matt Curtin, *Address Munging Considered Harmful* (http://www.interhack.net/pubs/munging-harmful/)

12. *WPoison* (http://www.monkeys.com/wpoison/)

13. Devin Carraway, *Sugarplum – spam poision* (http://www.devin.com/sugarplum/)

14. *Teegrubing FAQ* (http://www.iks-jena.de/mitarb/lutz/usenet/teergrube.en.html)

15. Vipul Ved Prakash, *Vipul's Razor* (http://razor.sourceforge.net/)

16. Rhyolite Software – *Distributed Checksum Clearinghouse* (http://www.rhyolite.com/anti-spam/dcc/)

17. *BrightMail Inc.* (http://www.brightmail.com)

18. Plaintiff's Complaint, Hartford House, Ltd. v. Microsoft Corp. (http://free.bluemountain.com/home/bluemountain_vs_Microsoft.html)

19. *SpamAssassin* (http://www.spamassassin.org)

20. *SpamAssassin – Tests Performed* (http://www.spamassassin.org/tests.html)

21. Bart De Win, *On the anonymity of electronic cash* (http://www.cs.kuleuven.ac.be/publicaties/rapporten/cw/CW316.pdf)

22. Cynthia Dwork and Moni Naor, *Pricing via Processing* (http://www.wisdom.weizmann.ac.il/~naor/onpub.html)

23. Adam Back - *Hash Cash* (http://www.cypherspace.org/~adam/hashcash/)

24. CAMRAM (CAMpaign for Real mAil) (http://www.camram.org)

25. D.J. Bernstein, *Internet Mail 2000* (http://cr.yp.to/im2000.html)

26. ProjectIM2000 wiki page (http://wiki.haribeau.de/cgi-bin/wiki.pl?ProjectIM2000)

27. Additional information on legal approaches to preventing spam is available at the *Spam Laws* web site (http://www.spamlaws.com) and David E. Sorkin, *Technical and Legal Approaches to Unsolicited Electronic Mail*, 35 U.S.F. L. Rev. 325 (2001).

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

Unix and Undergraduate Teaching

Dr Carlo Kopp
SCSSE
Monash University
carlo@mail.csse.monash.edu.au

Introduction

Unix user and basic administration skills are an essential component of the required skills package of a university Computer Sciences graduate. While many industry players have repeatedly predicted the demise of Unix based environments in recent years, the converse appears to be the case. In networking environments, development environments and large server environments, Unix remains the operating system of choice. With the emergence over the last decade of unencumbered Unix clones, be it of the Linux or *BSD varieties, supported by an immense pool of GPL applications, the Unix environment is likely to remain a core feature of computing environments for the foreseeable future.

Historically university Computer Science departments and schools have been a key source of early Unix skills training. Undergraduate students frequently acquired their earliest programming skills at university, on a variety of Unix platforms. Students acquired proficiency in the use of shells, compilers, debuggers, libraries and the X11 GUI environment.

If we look back a decade or two, the student population entering the university envirnment would have typically had minimal apriori computing skills. For all practical purposes they would more than often fit the 'tabula rasa' model - with no prior experience they could be taught from the ground up with an appropriate skills set, building progressively upon earlier subjects until they developed a good understanding of the environment.

Over the last decade we have seen some very significant changes, all of which are a consequence of 'commodification' in the computing world. Developed post-industrial nations, of which Australia is a good example, are characterised by a very high level of private computer ownership in households. With the cost of a very respectable desktop machine typically sitting between AU$1000.00 and AU$3000.00, most households from which university undergraduates are recruited will own one or more desktop computers.

Concurrently, commodification of hardware has seen large scale proliferation of desktop systems into secondary education establishments - most high schools will have at least one or more computing labs for students to learn on.

The consequence of commodification is that the profile of the typical first year university undergraduate has changed - the student may have several years of experience using and often administering a Windows/Intel system, or less frequently an Apple/PowerPC system - and very infrequently a Linux system. Such undergraduate students will enter university from an environment where they were previously regarded to be highly computer literate if not expert, and will perceive their own skills to be of a high standard, and highly valuable. More than often such students will take pride in their Windows/Intel or Apple/PowerPC skills set and actively promote the 'virtues' of both of these systems to their less skilled peers.

What this means in practical terms is that many students will be reluctant to learn another operating system and programming environment, as from their perspective it might mean relinquishing their status as 'experts' in their peer group and then having to compete on a truly level playing field.

Why Retain *nix as a Teaching Platform?

An argument which is frequently seen in academia is whether Unix and its derivatives should be retained as the primary platform for teaching programming and system administration skills. The counter-Unix case is often presented in terms of 'more than 90% of computers in the industry environment are Microsoft systems, therefore it makes sense to use Microsoft platforms as the basic platform'.

This argument has not been particularly successful, especially in the more established and mature university Computer Science departments and schools. There are a multiplicity of good reasons why Unix and its derivatives

should and indeed usually are retained as the primary teaching platform:

Cost (1): the advent of Linux and *BSD permits the deployment of large numbers of on campus Unix derivative systems at very low cost using cheap commodity hardware.

Cost (2): the advent of Linux and *BSD provides students with very cheap systems for working at home, with large packages of very competitive GPL tools, especially for software development. Dual booting permits these installations to coexist on home computers which are shared by the student with other family emembers.

Portability: an ongoing issue with many Microsoft tools is poor portability of source files between different releases of the same platform. As result, unless the on campus systems and students' home systems are of exactly the same configuration, this produces incessant difficulties with moving work from the comapus home and vice versa.

Performance: university labs and students' home machines are frequently 'hand me downs' which are short on memory, CPU clock speed and disk capacity. If a student has access to a machine with only 16 or 32 MB of memory, a Linux or *BSD installation will allow much more work to be done with a given set of computing resources.

Functionality: almost all Linux and *BSD releases are provided with a very rich package of GPL tools, especially development tools, on the CD-ROM or web accessible ISO images. Therefore the overheads in time (and dollar outlay) to provide a powerful development environment with C, C++, Java, Perl, LaTeX and other languages are minimised.

Source Code: GPL operating systems and tools are provided with source code. This is especially valuable for teaching environments as it provides opportunities for student projects in which existing GPL software can be modified, enhanced or indeed developed from scratch. Students get the opportunity to see tools as software with lines of code rather than black box entities provided by vendors.

Depth: While GPL operating systems now provide powerful GUI desktop environments, they retain access to the underlying environment. This permits debugging techniques which are not practical in binary only proprietary GUI environments. In turn this provides students with useful practical administration and debugging skills.

These are compelling pragmatic reasons for the retention of Unix derivative operating systems as teaching platforms. This is, however, not the only reason why Unix derivatives should remain the central teaching platform.

The second major consideration is the placement of graduates once they complete their University courses. For the forseeable future, Unix derivatives will continue to dominate as platforms for RDBMS systems, large multiuser commercial systems, web servers, web caches, NFS servers and engineering/scientific number crunching platforms. Graduates with a skills package which is deficient in Unix skills are implicitly at a competitive disadvantage against their peers who have strong Unix skills, when seeking employment in many key sectors of the industry. Unless the graduate is specifically aiming for employment as a supporter of or developer on a desktop system, such as Windows, the graduate is competing at a distinct disadvantage. Only a minority of employers are likely to make the substantial investment required to push a graduate up the Unix learning curve to a level of useful proficiency.

The changing workforce environment where a stronger emphasis is placed by many employers on self-training and prior experience held by employees creates a strong imperative for universities to equip their graduates with durable skills sets. Unix skills, like fundamental theory in the discipline, fall very much into this category.

Challenges in Teaching *nix

Teaching a student population, a large portion of whom have been previously indoctrinated to use proprietary desktop operating systems, presents some very interesting challenges for Computer Science educators.

Key issues may be summarised thus:

1. Overcoming fear of the unknown, and prior indoctrination with the belief that *nix is intractably difficult to use and administer.

2. Overcoming active resistance by a small minority who are convinced that *nix skills are of no value in the workforce.

3. Facilitating students who have an interest in running Linux or *BSD on their home systems.

4. Weaning students away from the exclusive use of point/click and menu interfaces to shell oriented user interfaces.

The best strategy for overcoming fear of Unix, in the author's experience, has been the gradual introduction to Unix environments. Rather than assault the student with a massive deluge of new information to be absorbed, a problem in its own right for new arrivals in the university system, students are provided with Unix exposure in non-programming subjects - the Unix shell and X11 environment are used as a platform for running other tools, such as document preparation toolsets. Typically two semesters worth of *nix exposure seems to be adequate to dispel most of the anxiety which undergraduates often experience with *nix. A very useful measure is the provision of a 'primer' website explaining why *nix skills are valuable, with links to relevant websites.

Active resistance against *nix is seen from time to time with a small minority of students. Typically such students will lobby heads of departments, associate deans or subdeans with petitions to exclude *nix from their subject syllabus, if they do not get any sympathy from the academic teaching the subject. Perhaps the only recipe for dealing with such individuals is patience, since reasoned argument about their future prospects in industry without *nix skills seldom strikes a chord with such an audience[1].

Facilitating access to *nix for student home system installations might appear to be trivial, but in practice it is often a challenge to organise. Many students are not accustomed to working independently and even the task of locating http://www.linux.org/ and acquiring an installation CD-ROM can present them with difficulties. Three strategies have proven to be useful for dealing with this problem. The first is the provision of a webpage with links to sites such as http://www.linux.org/, http://www.kde.org/, http://www.freebsd.org/, http://www.netbsd.org/ and other sites of interest. Another strategy is handing out free of charge Linux installation CD-ROMs - where the opportunity exists a box full of CD-ROMs at the main administration desk for the school or department is always helpful. A third approach exists where the students may have organised a 'Computer Club' or similar enthusiasts' grouping or body - a annual feature at Monash SCSSE is the 'Linux Install Night' where Student Club members volunteer time to install Linux for their less experienced peers.

Weaning students away from the exclusive use of GUIs can also present interesting problems. The strength of the X11 desktop and xterm is that it provides a relatively seamless transition between the two paradigms. Nevertheless, the author has had repeated instances of students using the *vim* (*vi* clone) text editor menus in preference to the use of standard command set syntax, even many weeks after introduction to the new environment.

While the problem of teaching GUI centric students to use shells can be formidable, other problems have also been observed. The most notable is a propensity for the more adventurous students to compete in terms of who can set up and configure the most esoteric desktop environment or indeed configuration thereof. Some students invest more effort into playing with KDE or Gnome than they invest in their planned laboratory tasks, to the detriment of the end product and the students' grades.

Conclusions

The commodification of computing hardware and large scale penetration of proprietray desktops into community households has been a mixed blessing for computing educators. While students arriving at universities may have prior experience, much if not all of that experience will be confined to proprietary desktop environments.

The challenges in transitioning to *nix those students who have been previously exposed only to proprietary desktop systems are not insurmountable, but the task does require a systematic effort and some careful thought.

A key aspect of any such effort is gradual, incremental exposure to the *nix environment, to dispel anxieties resulting from a total lack of prior experience, and from aggressive proprietory vendor marketing.

A second important aspect is facilitating easy access to Linux or *BSD installation CD-ROMs. Guidance by more experienced students who have existing household Linux or *BSD installations has proven to be invaluable in the task of increasing the number of student Linux or *BSD installations.

Industry can further facilitate this process by actively supporting university Computer Science schools and departments by making personnel available as guest lecturers on industry issues and the value of Unix skills, and by the provision of free Linux, *BSD or Unix installation CD-ROMs.

Unix and its derivatives remain a core technology for the industry and therefore Unix centric skills will remain an important component of a Computer Science or Software Engineering graduates' skills set. It is therefore important that this skills set is actively supported by both universities and the industry.

The Author

Born in Perth, Western Australia, Carlo Kopp graduated with first class honours in Electrical Engineering in 1984, from the University of Western Australia. In 1996 he completed an MSc in Computer Science and in 2000 a PhD in the same discipline, at Monash University in Melbourne.

[1]Some instances the author has dealt with would appear to be motivated by factors such as the student wishing to increase his peer group standing by visibly rebelling against authority, or a simple intent to minimise workload by avoiding new learning tasks.

Carlo has over a decade of diverse industry experience, including the design of high speed communications equipment, optical fibre receivers and transmitters, communications equipment including embedded code, Unix / SPARC computer workstation motherboards, Sbus graphics adaptors and chassis. More recently, he has consulted in Unix systems programming, performance engineering and system administration. His most notable industry project was the design of the first SPARC workstation motherboards to be commercially manufactured in Australia, in 1993.

Actively publishing in Australia's then leading Unix industry journal, AUUR/Systems/Systems Developer, between 1994 and 2002, Carlo specialised in writing technology reviews and fundamentals tutorials. He now lectures in Computer Architecture and Real Time Systems Design at Monash University.

Web page: `http://www.csse.monash.edu.au/~carlo/`

Extending the Wired LAN: 802.11

Adam Radford
System Engineer
Cisco Systems

Introduction

Wireless local area network (WLAN) technology is more secure, robust, and affordable than ever before. That's why more enterprises are taking advantage of the mobility and productivity benefits that WLANs provide.

However, WLAN success is not as simple as plugging in an access point (a wireless "hub") and handing out wireless cards. It requires careful consideration of your network and user requirements, preparation, and a sound deployment and security strategy.

The Essential Questions

Knowing the answer to the following question is the first step to a successful deployment: Why is the organization considering wireless? If you can clearly define the requirements of your wireless network, you can develop a deployment plan that will not only save networking dollars, but increase both productivity and user satisfaction.

How many of your users require mobility, and where do they need to go? What user applications will run over the WLAN? Listing the applications required by users will help to determine minimum bandwidth requirements and identify WLAN candidates. Keep in mind, however, that wireless is a shared medium, not a switched medium. While most mainstream networked applications can be migrated to a shared WLAN, it's not necessarily appropriate for all applications.

Knowing your users and their application requirements will help you to define coverage areas that don't waste money – or compromise security – by sending signals beyond the intended areas.

Choosing the Right Technology

After you define the requirements of your users, you will need to choose a wireless technology. For most enterprises, the choice is between IEEE standards 802.11b and 802.11a. The 802.11b standard has been widely adopted by vendors and customers who find its 11-Mbps data rate more than adequate for their applications. Interoperability between many of the products on the market is ensured through the Wireless Ethernet Compatibility Alliance (WECA) Wi-Fi certification program. Therefore, if your network requirements include supporting a wide variety of devices from different vendors, 802.11b is probably your best choice.

The higher data rate of the 802.11a standard (up to 54 Mbps) provides higher per-user throughput when the shared medium is properly provisioned. However, as of this writing, WECA testing for interoperability of 802.11a devices has not yet begun. Only a handful of companies are currently manufacturing 802.11a products (mainly access points and CardBus client adapters). Therefore, 802.11a systems should be targeted for end users whose applications require the higher bandwidth that the new standard affords. For more information on wireless standards, and to determine which one is right for your deployment, see "Welcome to the Wireless Enterprise."

Data Rates

Because data rates affect range, selecting data rates during the design stage is extremely important. Access points offer clients multiple data rates for the wireless link. For 802.11b, the range is from 1 to 11 Mbps in four increments, while for 802.11a the range is 6 to 54 Mbps in seven increments. The client cards will automatically switch to the fastest possible rate of the access point; how this is done varies from vendor to vendor. Because each data rate has

a unique cell of coverage (the higher the data rate, the smaller the cell), the minimum data rate for any given cell must be determined at the design stage. Cell sizes at given data rates can be thought of as being concentric circles with higher data-rate circles nested within the coverage area of the immediately higher data rate. Selecting only the highest data rate will require a greater number of access points to cover a given area; therefore, care must be taken to develop a compromise between required aggregate data rate and overall system cost.

Access Point Placement and Power

For a typical office, education, or health-care environment, access points can be mounted at ceiling height. For warehouses and other facilities with high ceilings, they should be mounted between 15 and 25 feet. Mounting access points at this height can create the additional problem of getting power to the unit. In these cases, you may want to consider a device that allows you to provide power over the Category 5 Ethernet cable – such as the Cisco Aironet 1200 Series, which can be powered by all Cisco line-power-enabled devices such as Catalyst switches, line-power patch panels. or a power injector that ships with the product. Powering access points in this manner can drastically reduce the cost and complication of installations.

In public areas such as retail stores, trying to keep the access point out of sight can create challenges as well. If it has to be placed above ceiling panels, you will typically want to place the antenna below the ceiling, where it will distribute the signal properly. This means you will need an access point with a remote antenna capability. Also, many regions legally require that devices installed above a ceiling be plenum rated, so be certain that the access point you choose has a metal casing that meets the specific fire code requirements of your area.

Antenna Selection and Placement

Each access point has a type of antenna system. Some, such as those designed for the 802.11a U-NII 1 indoor band, require an antenna that is permanently attached. Others have antennas that can be placed remotely via an antenna cable. However, because the coax cable used for RF has a very high signal loss, the antenna should not be mounted more than a few feet from the access point. The correct antenna can make the difference between a system that simply works and a system that works well. Proper antenna selection and placement also can drastically reduce the number of access points required, lowering overall system costs. For example, larger networks will benefit greatly if their design incorporates a 2.4-GHz external antenna.

Diversity antenna systems use two separate antennas or, in some cases, two antennas in a single enclosure. The radio then has the capability to choose the best antenna for receiving the signal. Using diversity antennas to overcome multipath interference, which occurs when radio waves bounce off objects and take multiple paths to their destination, can drastically improve the reception and size of your coverage area.

While the size of the coverage area is the most important determining factor for antenna selection and placement, it isn't the sole criterion. Building construction, ceiling height, internal obstructions, available mounting locations, and physical aesthetics all must be considered.

Cement and steel construction have different radio propagation characteristics and are therefore factors when determining antenna strength. Internal obstructions, such as product inventory and racking in warehouse environments, are also factors. For example, any product with highwater content will absorb 2.4-GHz RF energy. In a warehouse, shelves stacked with paper or cardboard products can, due to their high water content, create RF "shadows" or dead spots.

External considerations, such as public locations that prohibit the use of larger antennas, may require creative antenna placement so that they are unobtrusive and blend well with the surroundings.

Connecting to the Wired LAN

How will your WLAN connect to the wired network? Remember, the client will be moving from office to office, floor to floor, or maybe even building to building. If users are moving across subnets, you have new challenges to consider. Some operating systems (such as Windows XP and Windows 2000) support automatic Dynamic Host Control Protocol (DHCP) release/renew to obtain the IP address for the new subnet. However, certain IP applications such as virtual private networks (VPNs) will fail when this feature is enabled. This issue can be solved by deploying a flat network design for your WLANs, where all access points in a roaming area are on the same segment.

Many large companies use several flat networks. They determine where users will typically roam and try to segment the wireless network based on coverage areas with a minimum number of users roaming between them. If you don't have coverage outdoors, you'll lose IP connectivity as you move between buildings, so a good segmentation plan is to provide one subnet per building.

The Site Survey

A comprehensive site survey will help you define your coverage area and data rates, and determine the most precise placement of your access points. A prerequisite to the survey is to obtain as much information about the site as possible. Surveying involves both diagramming the coverage area and measuring the strength of the signal. While signal strength can tell you if the signal is strong enough to be received, signal-to-noise ratio (SNR) measurements help you compare the signal to the noise floor. If noise in the band is high enough, it can cause reception problems, even if there is a strong signal from the access point. Because signal strength alone is not sufficient, using both SNR measurements and packet retry count – the number of times packets had to be retransmitted for successful reception – is the best way to validate your coverage area.

Packet retry count, which should be below 10 percent in all areas, is the ultimate method for determining the edge of RF data reception. You may have areas where the signal is strong, but because of noise floor, or multipath interference, you cannot decode the signal, and the packet retry count will increase. However, without an SNR reading, you will not know whether packet retries are increasing because you are out of range, the noise is too high, or the signal is too low.

Your site survey must also determine if RF interference exists. An 802.11b WLAN uses the 2.4-GHz band, while an 802.11a WLAN uses the U-NII 1 and U-NII 2 5-GHz bands. Both of these are shared, unlicensed bands. Because they use a shared spectrum, neglecting to take into consideration interference created by microwave ovens, cordless phones, satellite systems, and other RF devices such as RF lighting systems and neighboring WLANs, can seriously affect performance.

Conducting the Site Survey

While physically examining your site, mark up the building layout or site map to show all coverage areas. Starting with the building outline, mark the outside areas that need coverage. Identify possible obstructions for RF such as freezers, coolers, X-ray rooms, elevators, and micro- waves. Because metal is highly reflective for RF signals, collections of office equipment such as metal bookshelves and cabinets also constitute potential RF problem areas. Mark all possible obstructions.

A successful wireless deployment ensures adequate coverage where needed but minimizes the coverage that extends beyond the physical campus. Failing to limit RF coverage can result in the network extending needlessly beyond the facility, potentially exposing the network to unauthorized access (see "How to Build a Secure WLAN"). Surveying from "the outside in" helps ensure that your coverage area does not extend beyond the facility. To do this, place your access point in the outside corner of your proposed coverage area (position A in Figure 1). Moving inward, locate the edge of your coverage area (position B) using your survey method of choice – the preferred being a combination of SNR and packet retry count.

Now move the access point to position B. Since position B represented the coverage edge when the access point was in position A, with the access point at position B, position A is now the edge, receiving adequate coverage without RF spilling beyond the facility walls. Performing another survey at position B will allow you to determine if your coverage area includes more users than the desired maximum number per access point. If so, you will need to reduce the coverage area by using a smaller antenna or lowering the power level of the access point. After the entire site survey is completed, summarize your findings in a report for the installation team, including your building diagram with access point placement and cell structures, antenna choice, configuration parameters, power requirements, possible interference sources, and photographs of each location.

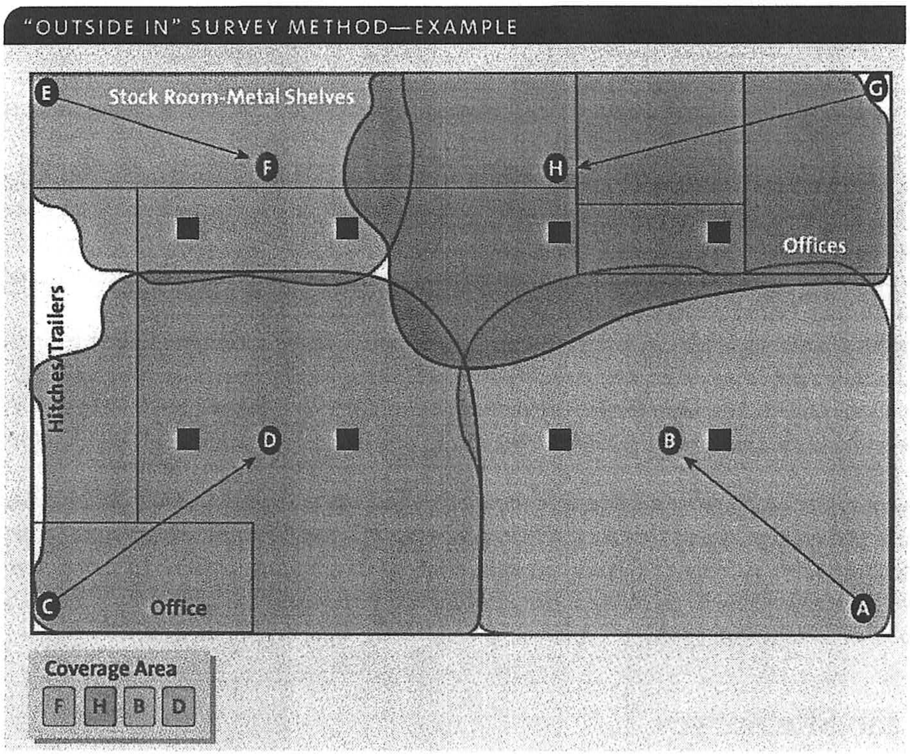
* * *

Without proper design and installation, a WLAN can be disappointing. Documenting user application and bandwidth requirements, choosing the appropriate technology and products, and completing a comprehensive site survey prior to installation will help ensure success and deliver a wireless network that meets or exceeds your – and your users' – greatest expectations.

Site Survey Checklist

Before beginning your site survey, make sure you have the following:

- Detailed layout of the building (that can be marked up), including where most users will be located.
- Portable battery pack or other method of powering access points.
- Description of the desired coverage areas and areas that do not need coverage.



- Number of users, descriptions of applications, and data rates, for determining how to properly provision your collision domains through access point placement.
- The same brand and model of WLAN equipment that will eventually be deployed.
- Antennas. Consider trying out more than one kind of antenna, because performance can vary in different coverage areas.
- Miscellaneous supplies:
  - Digital camera (to provide pictures for the installation team)
  - Tie wraps (for temporarily mounting access points and antennas)
  - Duct tape (also invaluable for temporarily mounting equipment)
  - Small flashlight (for seeing under ceiling tiles and the like)

UNIX and Open Source - The State of the Union

Mark White
co-founder and Managing Partner
Apviva Technology Partners
`http://www.apviva.com.au/`

In this presentation, Mark White continues the fine tradition of yearly retrospective and crystal ball gazing on the state of Unix and Open Source. A fitting way to close the conference and hopefully an opportunity for attendees to clarify their goals in their own area of the industry.

Mark has over 15 years of practical experience in the technology industry throughout Australia, South Asia, Korea, Greater China, India and the United States. His career has encompassed internationally focussed technical and marketing management roles with well-known US companies including Bell Atlantic International, Tandem Computers, Compaq Computer Corporation and Red Hat, for whom he successfully established and grew international operations as Vice-President and General Manager, Asia-Pacific.

A qualified software engineer and project manager, his career accomplishments also span marketing, business development, strategic planning and executive responsibility. His insight and hands-on experience throughout Asia and the United States allow him to assist technology companies in accelerating toward their full potential. He has a specific aptitude for working with early-stage and start-up companies and holds advisory positions with inQbator, DSTC and HCV Wireless.

Mark earned a BSc from the University of Queensland and holds postgraduate qualifications in International Management from the Queensland University of Technology.